Xilinx OpenCV User Guide

UG1233 (v2019.1) June 5, 2019

XILINX®

Revision History

The following table shows the revision history for this document.

Section	Revision Summary
06/05/2019 Version 2019.1	
General Updates	Minor editorial updates for 2019.1 release
Chapter 4: Getting Started with HLS	Added a new section
Chapter 3: Getting Started with SDAccel	Enhanced the existing functionality
Kalman Filter	Extended Kalman Filter support added
Color Conversion	Added additional color conversion formats
BoundingBox	Added a new function
Crop	Added a new function
xfOpenCV Library Functions	Added color image support in a few functions
xf::Mat Image Container Class	Minor updates
WarpTransform	Added the Warpaffine and Warpperspective support

Table of Contents

Chapter 1

Overview

This document describes the FPGA device optimized xfOpenCV library, called the Xilinx® xfOpenCV library and is intended for application developers using Zynq®-7000 SoC and Zynq® UltraScale+™ MPSoC and PCIE based (Virtex and U200 ...) devices. xfOpenCV library has been designed to work in the SDx™ development environment, and provides a software interface for computer vision functions accelerated on an FPGA device. xfOpenCV library functions are mostly similar in functionality to their OpenCV equivalent. Any deviations, if present, are documented.

***Note*:** For more information on the xfOpenCV library prerequisites, see the Prerequisites. To familiarize yourself with the steps required to use the xfOpenCV library functions, see the Using the xfOpenCV Library.

Basic Features

All xfOpenCV library functions follow a common format. The following properties hold true for all the functions.

- All the functions are designed as templates and all arguments that are images, must be provided as `xf::Mat`.
- All functions are defined in the `xf` namespace.
- Some of the major template arguments are:
  - Maximum size of the image to be processed
  - Datatype defining the properties of each pixel
  - Number of pixels to be processed per clock cycle
  - Other compile-time arguments relevent to the functionality.

The xfOpenCV library contains enumerated datatypes which enables you to configure `xf::Mat`. For more details on `xf::Mat`, see the xf::Mat Image Container Class.

xfOpenCV Kernel on the reVISION Platform

The xfOpenCV library is designed to be used with the SDx development environment. xfOpenCV kernels are evaluated on the reVISION platform.

The following steps describe the general flow of an example design, where both the input and the output are image files.

1. Read the image using `cv::imread()`.
2. Copy the data to `xf::Mat`.
3. Call the processing function(s) in xfOpenCV.
4. Copy the data from `xf::Mat` to `cv::Mat`.
5. Write the output to image using `cv::imwrite()`.

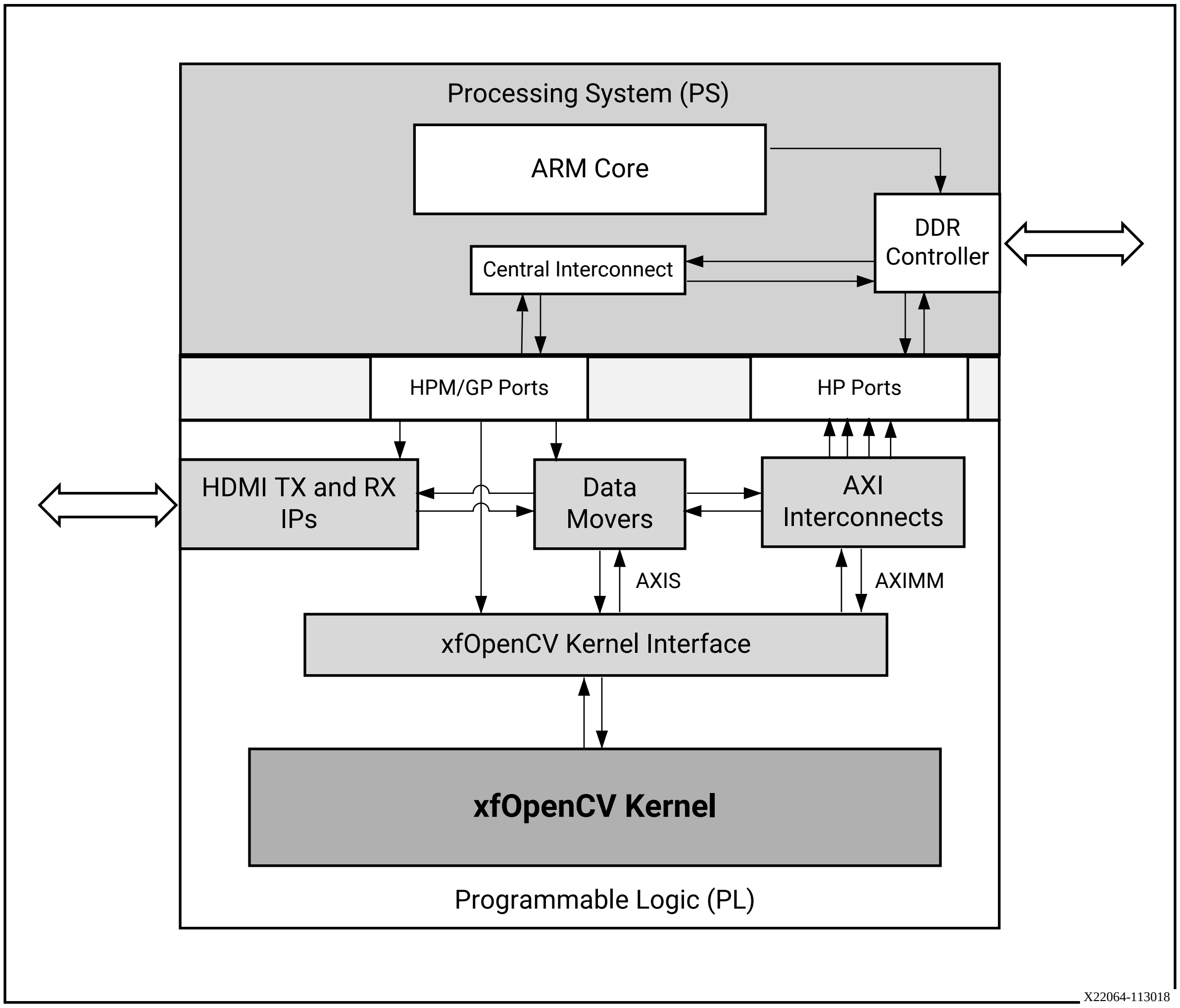
The entire code is written as the host code for the pipeline , from which all the calls to xfOpenCV functions are moved to hardware. Functions from xfOpenCV are used to read and write images in the memory. The image containers for xfOpenCV library functions are `xf::Mat` objects. For more information, see the xf::Mat Image Container Class.

The reVISION platform supports both live and file input-output (I/O) modes. For more details, see the reVISION Getting Started Guide.

- File I/O mode enables the controller to transfer images from SD Card to the hardware kernel. The following steps describe the file I/O mode.
  1. Processing system (PS) reads the image frame from the SD Card and stores it in the DRAM.
  2. The xfOpenCV kernel reads the image from the DRAM, processes it and stores the output back in the DRAM memory.
  3. The PS reads the output image frame from the DRAM and writes it back to the SD Card.
- Live I/O mode enables streaming frames into the platform, processing frames with the xfOpenCV kernel, and streaming out the frames through the appropriate interface. The following steps describe the live I/O mode.
  1. Video capture IPs receive a frame and store it in the DRAM.
  2. The xfOpenCV kernel fetches the image from the DRAM, processes the image, and stores the output in the DRAM.
  3. Display IPs read the output frame from the DRAM and transmits the frame through the appropriate display interface.

Following figure shows the reVISION platform with the xfOpenCV kernel block:

Figure 1: **xfOpenCV Kernel on the reVISION Platform**



***Note*:** For more information on the PS-PL interfaces and PL-DDR interfaces, see the *Zynq UltraScale+ Device Technical Reference Manual* (UG1085).

xfOpenCV Library Contents

The following table lists the contents of the xfOpenCV library.

Table 1: **xfOpenCV Library Contents**

Folder	Details
`include`	Contains the header files required by the library.
`include/common`	Contains the common library infrastructure headers, such as types specific to the library.
`include/core`	Contains the core library functionality headers, such as the `math` functions.
`include/features`	Contains the feature extraction kernel function definitions. For example, `Harris`.
`include/imgproc`	Contains all the kernel function definitions, except the ones available in the `features` folder.
`include/video`	Contains all the kernel function definitions, except the ones available in the `features and imgproc` folder.
`examples`	Contains the sample test bench code to facilitate running unit tests. The `examples/` folder contains the folders with algorithm names. Each algorithm folder contains host files, `.json` file, and `data` folder. For more details on how to use the xfOpenCV library, see xfOpenCV Kernel on the reVISION Platform.
`examples`	Contains the sample test bench code for 24 functions, which shows how to use xfOpenCV library in SDAccel™ environment.
`HLS_Use_Model`	Contains examples for using xfOpenCV functions in Standalone Vivado HLS in 2 different modes.
`HLS_Use_Model/Standalone_HLS_Example`	Contains sample code and tcl script for synthesizing xfOpenCV functions as is, in Standalone Vivado HLS tool.
`HLS_Use_Model/Standalone_HLS_AXI_Example`	Contains sample code and tcl script for synthesizing functions with AXI interfaces, in Standalone Vivado HLS tool.

Chapter 2

Getting Started with SDSoC

This chapter provides the information you need to bring up your design using the xfOpenCV library functions.

Prerequisites

This section lists the prerequisites for using the xfOpenCV library functions on ZCU104 based platforms. The methodology holds true for ZC702 and ZC706 reVISION platforms as well.

- Download and install the SDx development environment according to the directions provided in *SDSoC Environments Release Notes, Installation, and Licensing Guide* (UG1294). Before launching the SDx development environment on Linux, set the `$SYSROOT` environment variable to point to the Linux root file system if using terminal to build project, delivered with the reVISION platform. For example:

```
export SYSROOT = <local folder>/zcu104_rv_ss/sw/a53_linux/a53_linux/
sysroot/aarch64-xilinx-xilinx
```

- Download the Zynq® UltraScale+™ MPSoC Embedded Vision Platform zip file and extract its contents. Create the SDx development environment workspace in the `zcu104_rv_ss` folder of the extracted design file hierarchy. For more details, see the reVISION Getting Started Guide.
- Set up the ZCU104 evaluation board. For more details, see the reVISION Getting Started Guide.
- Download the xfOpenCV library. This library is made available through github. Run the following `git clone` command to clone the xfOpenCV repository to your local disk:

```
git clone https://github.com/Xilinx/xfopencv.git
```

Migrating HLS Video Library to xfOpenCV

The HLS video library will soon be deprecated .All the functions and most of the infrastructure available in HLS video library are now available in xfOpenCV with their names changed and some modifications. These HLS video library functions ported to xfOpenCV support SDSoc build flow also.

This section provides the details on using the C++ video processing functions and the infrastructure present in HLS video library.

Infrastructure Functions and Classes

All the functions imported from HLS video library now take xf::Mat (in sync with xfOpenCV library) to represent image data instead of hls::Mat. The main difference between these two is that the hls::Mat uses hls::stream to store the data whereas xf::Mat uses a pointer. Therefore, hls:: Mat cannot be exactly replaced with xf::Mat for migrating.

Below table summarizes the differences between member functions of hls::Mat to xf::Mat.

Table 2: **Infrastructure Functions and Classes**

Member Function	hls::Mat (HLS Video lib)	Xf::Mat (xfOpenCV lib)
channels()	Returns the number of channels	Returns the number of channels
type()	Returns the enum value of pixel type	Returns the enum value of pixel type
depth()	Returns the enum value of pixel type	Returns the depth of pixel including channels
read()	Readout a value and return it as a scalar from stream	Readout a value from a given location and return it as a packed (for multi-pixel/clock) value.
operator >>	Similar to read()	Not available in xfOpenCV
operator <<	Similar to write()	Not available in xfOpenCV
Write()	Write a scalar value into the stream	Writes a packed (for multi-pixel/clock) value into the given location..

Infrastructure files available in HLS Video Library hls_video_core.h, hls_video_mem.h, hls_video_types.h are moved to xf_video_core.h, xf_video_mem.h, xf_video_types.h in xfOpenCV Library and hls_video_imgbase.h is deprecated. Code inside these files unchanged except that these are now under xf::namespace.

Classes

- **Memory Window Buffer:** hls::window is now xf::window. No change in the implementation, except the namespace change. This is located in “xf_video_mem.h” file.

- **Memory Line Buffer:** hls::LineBuffer is now xf::LineBuffer. No difference between the two, except xf::LineBuffer has extra template arguments for inferring different types of RAM structures, for the storage structure used. Default storage type is "RAM_S2P_BRAM" with RESHAPE_FACTOR=1. Complete description can be found here xf::LineBuffer. This is located in `xf_video_mem.h` file.

Funtions

- **OpenCV interface functions:** These functions covert image data of OpenCV Mat format to/from HLS AXI types. HLS Video Library had 14 interface functions, out of which, two functions are available in xfOpenCV Library: cvMat2AXIvideo and AXIvideo2cvMat located in "xf_axi.h" file. The rest are all deprecated.
- **AXI4-Stream I/O Functions:** The I/O functions which convert hls::Mat to/from AXI4-Stream compatible data type (hls::stream) are hls::AXIvideo2Mat, hls::Mat2AXIvideo. These functions are now deprecated and added 2 new functions xf::AXIvideo2xfMat and xf:: xfMat2AXIvideo to facilitate the xf::Mat to/from conversion. To use these functions, the header file "xf_infra.h" must be included.

xf::window

A template class to represent the 2D window buffer. It has three parameters to specify the number of rows, columns in window buffer and the pixel data type.

Class definition

```
template<int ROWS, int COLS, typename T>
class Window {
public:
    Window()
   /* Window main APIs */
    void shift_pixels_left();
    void shift_pixels_right();
    void shift_pixels_up();
    void shift_pixels_down();
    void insert_pixel(T value, int row, int col);
    void insert_row(T value[COLS], int row);
    void insert_top_row(T value[COLS]);
    void insert_bottom_row(T value[COLS]);
    void insert_col(T value[ROWS], int col);
    void insert_left_col(T value[ROWS]);
    void insert_right_col(T value[ROWS]);
    T& getval(int row, int col);
    T& operator ()(int row, int col);
    T val[ROWS][COLS];
#ifdef __DEBUG__
    void restore_val();
    void window_print();
    T val_t[ROWS][COLS];
#endif
};
```

Parameter Descriptions

The following table lists the xf::Window class members and their descriptions.

Table 3: **Window Function Parameter Descriptions**

Parameter	Description
Val	2-D array to hold the contents of buffer.

Member Function Description

Table 4: **Member Function Description**

Function	Description
shift_pixels_left()	Shift the window left, that moves all stored data within the window right, leave the leftmost column (col = COLS-1) for inserting new data.
shift_pixels_right()	Shift the window right, that moves all stored data within the window left, leave the rightmost column (col = 0) for inserting new data.
shift_pixels_up()	Shift the window up, that moves all stored data within the window down, leave the top row (row = ROWS-1) for inserting new data.
shift_pixels_down()	Shift the window down, that moves all stored data within the window up, leave the bottom row (row = 0) for inserting new data.
insert_pixel(T value, int row, int col)	Insert a new element value at location (row, column) of the window.
insert_row(T value[COLS], int row)	Inserts a set of values in any row of the window.
insert_top_row(T value[COLS])	Inserts a set of values in the top row = 0 of the window.
insert_bottom_row(T value[COLS])	Inserts a set of values in the bottom row = ROWS-1 of the window.
insert_col(T value[ROWS], int col)	Inserts a set of values in any column of the window.
insert_left_col(T value[ROWS])	Inserts a set of values in left column = 0 of the window.
insert_right_col(T value[ROWS])	Inserts a set of values in right column = COLS-1 of the window.
T& getval(int row, int col)	Returns the data value in the window at position (row,column).
T& operator ()(int row, int col)	Returns the data value in the window at position (row,column).
restore_val()	Restore the contents of window buffer to another array.
window_print()	Print all the data present in window buffer onto console.

Template Parameter Description

Table 5: **Template Parameter Description**

Parameter	Description
ROWS	Number of rows in the window buffer.

Table 5: **Template Parameter Description** *(cont'd)*

Parameter	Description
COLS	Number of columns in the window buffer.
T	Data type of pixel in the window buffer.

Sample code for window buffer declaration

```
Window<K_ROWS, K_COLS, unsigned char> kernel;
```

xf::LineBuffer

A template class to represent 2D line buffer. It has three parameters to specify the number of rows, columns in window buffer and the pixel data type.

Class definition

```
template<int ROWS, int COLS, typename T, XF_ramtype_e
MEM_TYPE=RAM_S2P_BRAM, int RESHAPE_FACTOR=1>
 class LineBuffer {
public:
    LineBuffer()
        /* LineBuffer main APIs */
    /* LineBuffer main APIs */
    void shift_pixels_up(int col);
    void shift_pixels_down(int col);
    void insert_bottom_row(T value, int col);
    void insert_top_row(T value, int col);
    void get_col(T value[ROWS], int col);
    T& getval(int row, int col);
    T& operator ()(int row, int col);

    /* Back compatible APIs */
    void shift_up(int col);
    void shift_down(int col);
    void insert_bottom(T value, int col);
    void insert_top(T value, int col);
    T val[ROWS][COLS];
#ifdef __DEBUG__
    void restore_val();
    void linebuffer_print(int col);
    T val_t[ROWS][COLS];
#endif
};
```

Parameter Descriptions

The following table lists the xf::LineBuffer class members and their descriptions.

Table 6: **Line Buffer Function Parameter Descriptions**

Parameter	Description
Val	2-D array to hold the contents of line buffer.

Member Functions Description

Table 7: **Member Functions Description**

Function	Description
shift_pixels_up(int col)	Line buffer contents Shift up, new values will be placed in the bottom row=ROWS-1.
shift_pixels_down(int col)	Line buffer contents Shift down, new values will be placed in the top row=0.
insert_bottom_row(T value, int col)	Inserts a new value in bottom row= ROWS-1 of the line buffer.
insert_top_row(T value, int col)	Inserts a new value in top row=0 of the line buffer.
get_col(T value[ROWS], int col)	Get a column value of the line buffer.
T& getval(int row, int col)	Returns the data value in the line buffer at position (row, column).
T& operator ()(int row, int col);	Returns the data value in the line buffer at position (row, column).

Template Parameter Description

Table 8: **Template Parameter Description**

Parameter	Description
ROWS	Number of rows in line buffer.
COLS	Number of columns in line buffer.
T	Data type of pixel in line buffer.
MEM_TYPE	Type of storage element. It takes one of the following enumerated values: RAM_1P_BRAM, RAM_1P_URAM, RAM_2P_BRAM, RAM_2P_URAM, RAM_S2P_BRAM, RAM_S2P_URAM, RAM_T2P_BRAM, RAM_T2P_URAM.
RESHAPE_FACTOR	Specifies the amount to divide an array.

Sample code for line buffer declaration:

```
LineBuffer<3, 1920, XF_8UC3, RAM_S2P_URAM,1>     buff;
```

Video Processing Functions

The following table summarizes the video processing functions ported from HLS Video Library into xfOpenCV Library along with the API modifications.

Table 9: **Video Processing Functions**

Functions	HLS Video Library -API	xfOpenCV Library-API

addS	`template<int ROWS, int COLS, int SRC_T,`
	`typename _T, int DST_T>`
	`void AddS(Mat<ROWS, COLS,`
	`SRC_T>&src,Scalar<HLS_MAT_CN(SRC_T),`
	`_T> scl, Mat<ROWS, COLS, DST_T>& dst)`
		`template<int POLICY_TYPE, int SRC_T,`
		`int ROWS, int COLS, int NPC =1>`
		`void addS(xf::Mat<SRC_T, ROWS, COLS,`
		`NPC> & _src1, unsigned char`
		`_scl[XF_CHANNELS(SRC_T,NPC)],xf::Mat<SRC`
		`_T, ROWS, COLS, NPC> & _dst)`

AddWeighted	`template<int ROWS, int COLS, int`
	`SRC1_T, int SRC2_T, int DST_T, typename`
	`P_T>`
	`void AddWeighted(Mat<ROWS, COLS,`
	`SRC1_T>& src1,P_T alpha,Mat<ROWS, COLS,`
	`SRC2_T>& src2,P_T beta, P_T`
	`gamma,Mat<ROWS, COLS, DST_T>& dst)`
		`template< int SRC_T,int DST_T, int`
		`ROWS, int COLS, int NPC = 1>`
		`void addWeighted(xf::Mat<SRC_T, ROWS,`
		`COLS, NPC> & src1,float alpha,`
		`xf::Mat<SRC_T, ROWS, COLS, NPC> &`
		`src2,float beta, float gama,`
		`xf::Mat<DST_T, ROWS, COLS, NPC> & dst)`

Cmp	`template<int ROWS, int COLS, int`
	`SRC1_T, int SRC2_T, int DST_T>`
	`void Cmp(Mat<ROWS, COLS, SRC1_T>&`
	`src1,Mat<ROWS, COLS, SRC2_T>& src2,`
	`Mat<ROWS, COLS, DST_T>& dst,int cmp_op)`
		`template<int CMP_OP, int SRC_T, int`
		`ROWS, int COLS, int NPC =1>`
		`void compare(xf::Mat<SRC_T, ROWS, COLS,`
		`NPC> & _src1, xf::Mat<SRC_T, ROWS,`
		`COLS, NPC> & _src2,xf::Mat<SRC_T, ROWS,`
		`COLS, NPC> & _dst)`

CmpS	`template<int ROWS, int COLS, int SRC_T,`
	`typename P_T, int DST_T>`
	`void CmpS(Mat<ROWS, COLS, SRC_T>& src,`
	`P_T value, Mat<ROWS, COLS, DST_T>& dst,`
	`int cmp_op)`
		`template<int CMP_OP, int SRC_T, int`
		`ROWS, int COLS, int NPC =1>`
		`void compare(xf::Mat<SRC_T, ROWS, COLS,`
		`NPC> & _src1, unsigned char`
		`_scl[XF_CHANNELS(SRC_T,NPC)],xf::Mat<SRC`
		`_T, ROWS, COLS, NPC> & _dst)`

Max	`template<int ROWS, int COLS, int`
	`SRC1_T, int SRC2_T, int DST_T>`
	`void Max(Mat<ROWS, COLS, SRC1_T>& src1,`
	`        Mat<ROWS, COLS, SRC2_T>& src2,`
	`        Mat<ROWS, COLS, DST_T>& dst)`
		`template<int SRC_T, int ROWS, int COLS,`
		`int NPC =1>`
		`void Max(xf::Mat<SRC_T, ROWS, COLS,`
		`NPC> & _src1, xf::Mat<SRC_T, ROWS,`
		`COLS, NPC> & _src2,xf::Mat<SRC_T, ROWS,`
		`COLS, NPC> & _dst)`

MaxS	`template<int ROWS, int COLS, int SRC_T,`
	`typename _T, int DST_T>`
	`void MaxS(Mat<ROWS, COLS, SRC_T>& src,`
	`_T value, Mat<ROWS, COLS, DST_T>& dst)`
		`template< int SRC_T, int ROWS, int`
		`COLS, int NPC =1>`
		`void max(xf::Mat<SRC_T, ROWS, COLS,`
		`NPC> & _src1,  unsigned char`
		`_scl[XF_CHANNELS(SRC_T,NPC)],xf::Mat<SRC`
		`_T, ROWS, COLS, NPC> & _dst)`

Table 9: **Video Processing Functions** *(cont'd)*

Functions	HLS Video Library -API	xfOpenCV Library-API
Min	template<int ROWS, int COLS, int SRC1_T, int SRC2_T, int DST_T> void Min(Mat<ROWS, COLS, SRC1_T>& src1, Mat<ROWS, COLS, SRC2_T>& src2, Mat<ROWS, COLS, DST_T>& dst)	template< int SRC_T, int ROWS, int COLS, int NPC =1> void Min(xf::Mat<SRC_T, ROWS, COLS, NPC> & _src1, xf::Mat<SRC_T, ROWS, COLS, NPC> & _src2,xf::Mat<SRC_T, ROWS, COLS, NPC> & _dst)
MinS	template<int ROWS, int COLS, int SRC_T, typename _T, int DST_T> void MinS(Mat<ROWS, COLS, SRC_T>& src, _T value,Mat<ROWS, COLS, DST_T>& dst)	template< int SRC_T, int ROWS, int COLS, int NPC =1> void min(xf::Mat<SRC_T, ROWS, COLS, NPC> & _src1, unsigned char _scl[XF_CHANNELS(SRC_T,NPC)],xf::Mat<SRC_T, ROWS, COLS, NPC> & _dst)
PaintMask	template<int SRC_T,int MASK_T,int ROWS,int COLS> void PaintMask(Mat<ROWS,COLS,SRC_T> &_src, Mat<ROWS,COLS,MASK_T>&_mask, Mat<ROWS,COLS,SRC_T>&_dst,Scalar<HLS_MAT_CN(SRC_T),HLS_TNAME(SRC_T)> _color)	template< int SRC_T,int MASK_T, int ROWS, int COLS,int NPC=1> void paintmask(xf::Mat<SRC_T, ROWS, COLS, NPC> & _src_mat, xf::Mat<MASK_T, ROWS, COLS, NPC> & in_mask, xf::Mat<SRC_T, ROWS, COLS, NPC> & _dst_mat, unsigned char _color[XF_CHANNELS(SRC_T,NPC)])
Reduce	template<typename INTER_SUM_T, int ROWS, int COLS, int SRC_T, int DST_ROWS, int DST_COLS, int DST_T> void Reduce(Mat<ROWS, COLS, SRC_T> &src, Mat<DST_ROWS, DST_COLS, DST_T> &dst, int dim, int op=HLS_REDUCE_SUM)	template< int REDUCE_OP, int SRC_T,int DST_T, int ROWS, int COLS,int ONE_D_HEIGHT, int ONE_D_WIDTH, int NPC=1> void reduce(xf::Mat<SRC_T, ROWS, COLS, NPC> & _src_mat, xf::Mat<DST_T, ONE_D_HEIGHT, ONE_D_WIDTH, 1> & _dst_mat, unsigned char dim)
Zero	template<int ROWS, int COLS, int SRC_T, int DST_T> void Zero(Mat<ROWS, COLS, SRC_T>& src, Mat<ROWS, COLS, DST_T>& dst)	template< int SRC_T, int ROWS, int COLS, int NPC =1> void zero(xf::Mat<SRC_T, ROWS, COLS, NPC> & _src1,xf::Mat<SRC_T, ROWS, COLS, NPC> & _dst)
Sum	template<typename DST_T, int ROWS, int COLS, int SRC_T> Scalar<HLS_MAT_CN(SRC_T), DST_T> Sum(Mat<ROWS, COLS, SRC_T>& src)	template< int SRC_T, int ROWS, int COLS, int NPC = 1> void sum(xf::Mat<SRC_T, ROWS, COLS, NPC> & src1, double sum[XF_CHANNELS(SRC_T,NPC)])
SubS	template<int ROWS, int COLS, int SRC_T, typename _T, int DST_T> void SubS(Mat<ROWS, COLS, SRC_T>& src, Scalar<HLS_MAT_CN(SRC_T), _T> scl, Mat<ROWS, COLS, DST_T>& dst)	template<int POLICY_TYPE, int SRC_T, int ROWS, int COLS, int NPC =1> void SubS(xf::Mat<SRC_T, ROWS, COLS, NPC> & _src1, unsigned char _scl[XF_CHANNELS(SRC_T,NPC)],xf::Mat<SRC_T, ROWS, COLS, NPC> & _dst)

Table 9: **Video Processing Functions** *(cont'd)*

Functions	HLS Video Library -API	xfOpenCV Library-API
SubRS	template<int ROWS, int COLS, int SRC_T, typename _T, int DST_T> void SubRS(Mat<ROWS, COLS, SRC_T>& src, Scalar<HLS_MAT_CN(SRC_T), _T> scl, Mat<ROWS, COLS, DST_T>& dst)	template<int POLICY_TYPE, int SRC_T, int ROWS, int COLS, int NPC =1> void SubRS(xf::Mat<SRC_T, ROWS, COLS, NPC> & _src1, unsigned char _scl[XF_CHANNELS(SRC_T,NPC)],xf::Mat<SRC_T, ROWS, COLS, NPC> & _dst)
Set	template<int ROWS, int COLS, int SRC_T, typename _T, int DST_T> void Set(Mat<ROWS, COLS, SRC_T>& src, Scalar<HLS_MAT_CN(SRC_T), _T> scl, Mat<ROWS, COLS, DST_T>& dst)	template< int SRC_T, int ROWS, int COLS, int NPC =1> void set(xf::Mat<SRC_T, ROWS, COLS, NPC> & _src1, unsigned char _scl[XF_CHANNELS(SRC_T,NPC)],xf::Mat<SRC_T, ROWS, COLS, NPC> & _dst)
Absdiff	template<int ROWS, int COLS, int SRC1_T, int SRC2_T, int DST_T> void AbsDiff(Mat<ROWS, COLS, SRC1_T>& src1, Mat<ROWS, COLS, SRC2_T>& src2, Mat<ROWS, COLS, DST_T>& dst)	template<int SRC_T, int ROWS, int COLS, int NPC =1> void absdiff(xf::Mat<SRC_T, ROWS, COLS, NPC> & _src1,xf::Mat<SRC_T, ROWS, COLS, NPC> & _src2,xf::Mat<SRC_T, ROWS, COLS, NPC> & _dst)
And	template<int ROWS, int COLS, int SRC1_T, int SRC2_T, int DST_T> void And(Mat<ROWS, COLS, SRC1_T>& src1, Mat<ROWS, COLS, SRC2_T>& src2, Mat<ROWS, COLS, DST_T>& dst)	template<int SRC_T, int ROWS, int COLS, int NPC = 1> void bitwise_and(xf::Mat<SRC_T, ROWS, COLS, NPC> & _src1, xf::Mat<SRC_T, ROWS, COLS, NPC> & _src2, xf::Mat<SRC_T, ROWS, COLS, NPC> &_dst)
Dilate	template<int Shape_type,int ITERATIONS,int SRC_T, int DST_T, typename KN_T,int IMG_HEIGHT,int IMG_WIDTH,int K_HEIGHT,int K_WIDTH> void Dilate(Mat<IMG_HEIGHT, IMG_WIDTH, SRC_T>&_src,Mat<IMG_HEIGHT, IMG_WIDTH, DST_T&_dst,Window<K_HEIGHT,K_WIDTH,KN_T> &_kernel)	template<int BORDER_TYPE, int TYPE, int ROWS, int COLS,int K_SHAPE,int K_ROWS,int K_COLS, int ITERATIONS, int NPC=1> void dilate (xf::Mat<TYPE, ROWS, COLS, NPC> & _src, xf::Mat<TYPE, ROWS, COLS, NPC> & _dst,unsigned char _kernel[K_ROWS*K_COLS])
Duplicate	template<int ROWS, int COLS, int SRC_T, int DST_T> void Duplicate(Mat<ROWS, COLS, SRC_T>& src,Mat<ROWS, COLS, DST_T>& dst1,Mat<ROWS, COLS, DST_T>& dst2)	template<int SRC_T, int ROWS, int COLS,int NPC> void duplicateMat(xf::Mat<SRC_T, ROWS, COLS, NPC> & _src, xf::Mat<SRC_T, ROWS, COLS, NPC> & _dst1,xf::Mat<SRC_T, ROWS, COLS, NPC> & _dst2)
EqualizeHist	template<int SRC_T, int DST_T,int ROW, int COL> void EqualizeHist(Mat<ROW, COL, SRC_T>&_src,Mat<ROW, COL, DST_T>&_dst)	template<int SRC_T, int ROWS, int COLS, int NPC = 1> void equalizeHist(xf::Mat<SRC_T, ROWS, COLS, NPC> & _src,xf::Mat<SRC_T, ROWS, COLS, NPC> & _src1,xf::Mat<SRC_T, ROWS, COLS, NPC> & _dst)

Table 9: **Video Processing Functions** *(cont'd)*

Functions	HLS Video Library -API	xfOpenCV Library-API
erode	`template<int Shape_type,int ITERATIONS,int SRC_T, int DST_T, typename KN_T,int IMG_HEIGHT,int IMG_WIDTH,int K_HEIGHT,int K_WIDTH> void Erode(Mat<IMG_HEIGHT, IMG_WIDTH, SRC_T>&_src,Mat<IMG_HEIGHT,IMG_WIDTH,DST_T>&_dst,Window<K_HEIGHT,K_WIDTH,KN_T>&_kernel)`	`template<int BORDER_TYPE, int TYPE, int ROWS, int COLS,int K_SHAPE,int K_ROWS,int K_COLS, int ITERATIONS, int NPC=1> void erode (xf::Mat<TYPE, ROWS, COLS, NPC> & _src, xf::Mat<TYPE, ROWS, COLS, NPC> & _dst,unsigned char _kernel[K_ROWS*K_COLS])`
FASTX	`template<int SRC_T,int ROWS,int COLS> void FASTX(Mat<ROWS,COLS,SRC_T> &_src, Mat<ROWS,COLS,HLS_8UC1>&_mask,HLS_TNAME(SRC_T)_threshold,bool _nomax_supression)`	`template<int NMS,int SRC_T,int ROWS, int COLS,int NPC=1> void fast(xf::Mat<SRC_T, ROWS, COLS, NPC> & _src_mat,xf::Mat<SRC_T, ROWS, COLS, NPC> & _dst_mat,unsigned char _threshold)`
Filter2D	`template<int SRC_T, int DST_T, typename KN_T, typename POINT_T, int IMG_HEIGHT,int IMG_WIDTH,int K_HEIGHT,int K_WIDTH> void Filter2D(Mat<IMG_HEIGHT, IMG_WIDTH, SRC_T> &_src,Mat<IMG_HEIGHT, IMG_WIDTH, DST_T> &_dst,Window<K_HEIGHT,K_WIDTH,KN_T>&_kernel,Point_<POINT_T>anchor)`	`template<int BORDER_TYPE,int FILTER_WIDTH,int FILTER_HEIGHT, int SRC_T,int DST_T, int ROWS, int COLS,int NPC> void filter2D(xf::Mat<SRC_T, ROWS, COLS, NPC> & _src_mat,xf::Mat<DST_T, ROWS, COLS, NPC> & _dst_mat,short int filter[FILTER_HEIGHT*FILTER_WIDTH],unsigned char _shift)`
GaussianBlur	`template<int KH,int KW,typename BORDERMODE,int SRC_T,int DST_T,int ROWS,int COLS> void GaussianBlur(Mat<ROWS, COLS, SRC_T> &_src, Mat<ROWS, COLS, DST_T> &_dst,double sigmaX=0,double sigmaY=0)`	`template<int FILTER_SIZE, int BORDER_TYPE, int SRC_T, int ROWS, int COLS,int NPC = 1> void GaussianBlur(xf::Mat<SRC_T, ROWS, COLS, NPC> & _src, xf::Mat<SRC_T, ROWS, COLS, NPC> & _dst, float sigma)`
Harris	`template<int blockSize,int Ksize,typename KT,int SRC_T,int DST_T,int ROWS,int COLS> void Harris(Mat<ROWS, COLS, SRC_T> &_src,Mat<ROWS, COLS, DST_T>&_dst,KT k,int threshold`	`template<int FILTERSIZE,int BLOCKWIDTH, int NMSRADIUS,int SRC_T,int ROWS, int COLS,int NPC=1,bool USE_URAM=false> void cornerHarris(xf::Mat<SRC_T, ROWS, COLS, NPC> & src,xf::Mat<SRC_T, ROWS, COLS, NPC> & dst,uint16_t threshold, uint16_t k)`
CornerHarris	`template<int blockSize,int Ksize,typename KT,int SRC_T,int DST_T,int ROWS,int COLS> void CornerHarris( Mat<ROWS, COLS, SRC_T>&_src,Mat<ROWS, COLS, DST_T>&_dst,KT k)`	`template<int FILTERSIZE,int BLOCKWIDTH, int NMSRADIUS,int SRC_T,int ROWS, int COLS,int NPC=1,bool USE_URAM=false> void cornerHarris(xf::Mat<SRC_T, ROWS, COLS, NPC> & src,xf::Mat<SRC_T, ROWS, COLS, NPC> & dst,uint16_t threshold, uint16_t k`

Table 9: **Video Processing Functions** *(cont'd)*

Functions	HLS Video Library -API	xfOpenCV Library-API
HoughLines2	template<unsigned int theta,unsigned int rho,typename AT,typename RT,int SRC_T,int ROW,int COL,unsigned int linesMax> void HoughLines2(Mat<ROW,COL,SRC_T> &_src, Polar_<AT,RT> (&_lines) [linesMax],unsigned int threshold)	template<unsigned int RHO,unsigned int THETA,int MAXLINES,int DIAG,int MINTHETA,int MAXTHETA,int SRC_T, int ROWS, int COLS,int NPC> void HoughLines(xf::Mat<SRC_T, ROWS, COLS, NPC> & _src_mat,float outputrho[MAXLINES],float outputtheta[MAXLINES],short threshold,short linesmax)
Integral	template<int SRC_T, int DST_T, int ROWS,int COLS> void Integral(Mat<ROWS, COLS, SRC_T>&_src, Mat<ROWS+1, COLS+1, DST_T>&_sum)	template<int SRC_TYPE,int DST_TYPE, int ROWS, int COLS, int NPC> void integral(xf::Mat<SRC_TYPE, ROWS, COLS, NPC> & _src_mat, xf::Mat<DST_TYPE, ROWS, COLS, NPC> & _dst_mat)
Merge	template<int ROWS, int COLS, int SRC_T, int DST_T> void Merge(Mat<ROWS, COLS, SRC_T>& src0, Mat<ROWS, COLS, SRC_T>& src1, Mat<ROWS, COLS, SRC_T>& src2, Mat<ROWS, COLS, SRC_T>& src3, Mat<ROWS, COLS, DST_T>& dst)	template<int SRC_T, int DST_T, int ROWS, int COLS, int NPC=1> void merge(xf::Mat<SRC_T, ROWS, COLS, NPC> &_src1, xf::Mat<SRC_T, ROWS, COLS, NPC> &_src2, xf::Mat<SRC_T, ROWS, COLS, NPC> &_src3, xf::Mat<SRC_T, ROWS, COLS, NPC> &_src4, xf::Mat<DST_T, ROWS, COLS, NPC> &_dst)
MinMaxLoc	template<int ROWS, int COLS, int SRC_T, typename P_T> void MinMaxLoc(Mat<ROWS, COLS, SRC_T>& src, P_T* min_val,P_T* max_val,Point& min_loc, Point& max_loc)	template<int SRC_T,int ROWS,int COLS,int NPC=0> void minMaxLoc(xf::Mat<SRC_T, ROWS, COLS, NPC> & _src,int32_t *min_value, int32_t *max_value,uint16_t *_minlocx, uint16_t *_minlocy, uint16_t *_maxlocx, uint16_t *_maxlocy)
Mul	template<int ROWS, int COLS, int SRC1_T, int SRC2_T, int DST_T> void Mul(Mat<ROWS, COLS, SRC1_T>& src1, Mat<ROWS, COLS, SRC2_T>& src2, Mat<ROWS, COLS, DST_T>& dst)	template<int POLICY_TYPE, int SRC_T, int ROWS, int COLS, int NPC = 1> void multiply(xf::Mat<SRC_T, ROWS, COLS, NPC> & src1, xf::Mat<SRC_T, ROWS, COLS, NPC> & src2, xf::Mat<SRC_T, ROWS, COLS, NPC> & dst,float scale)
Not	template<int ROWS, int COLS, int SRC_T, int DST_T> void Not(Mat<ROWS, COLS, SRC_T>& src, Mat<ROWS, COLS, DST_T>& dst)	template<int SRC_T, int ROWS, int COLS, int NPC = 1> void bitwise_not(xf::Mat<SRC_T, ROWS, COLS, NPC> & src, xf::Mat<SRC_T, ROWS, COLS, NPC> & dst)
Range	template<int ROWS, int COLS, int SRC_T, int DST_T, typename P_T> void Range(Mat<ROWS, COLS, SRC_T>& src, Mat<ROWS, COLS, DST_T>& dst, P_T start,P_T end)	template<int SRC_T, int ROWS, int COLS,int NPC=1> void inRange(xf::Mat<SRC_T, ROWS, COLS, NPC> & src,unsigned char lower_thresh,unsigned char upper_thresh,xf::Mat<SRC_T, ROWS, COLS, NPC> & dst)

Table 9: **Video Processing Functions** *(cont'd)*

Functions	HLS Video Library -API	xfOpenCV Library-API

Resize	`template<int SRC_T, int ROWS,int COLS,int DROWS,int DCOLS>`
	`void Resize (`
	`Mat<ROWS, COLS, SRC_T> &_src,`
	`Mat<DROWS, DCOLS, SRC_T> &_dst,`
	`int interpolation=HLS_INTER_LINEAR )`
		`template<int INTERPOLATION_TYPE, int TYPE, int SRC_ROWS, int SRC_COLS, int DST_ROWS, int DST_COLS, int NPC, int MAX_DOWN_SCALE>`
		`void resize (xf::Mat<TYPE, SRC_ROWS, SRC_COLS, NPC> & _src, xf::Mat<TYPE, DST_ROWS, DST_COLS, NPC> & _dst)`

sobel	`template<int XORDER, int YORDER, int SIZE, int SRC_T, int DST_T, int ROWS,int COLS,int DROWS,int DCOLS>` `void Sobel (Mat<ROWS, COLS, SRC_T> &_src,Mat<DROWS, DCOLS, DST_T> &_dst)`	`template<int BORDER_TYPE,int FILTER_TYPE, int SRC_T,int DST_T, int ROWS, int COLS,int NPC=1,bool USE_URAM = false>` `void Sobel(xf::Mat<SRC_T, ROWS, COLS, NPC> & _src_mat,xf::Mat<DST_T, ROWS, COLS, NPC> & _dst_matx,xf::Mat<DST_T, ROWS, COLS, NPC> & _dst_maty)`

split	`template<int ROWS, int COLS, int SRC_T, int DST_T>`
	`void Split(`
	`Mat<ROWS, COLS, SRC_T>& src,`
	`Mat<ROWS, COLS, DST_T>& dst0,`
	`Mat<ROWS, COLS, DST_T>& dst1,`
	`Mat<ROWS, COLS, DST_T>& dst2,`
	`Mat<ROWS, COLS, DST_T>& dst3)`
		`template<int SRC_T, int DST_T, int ROWS, int COLS, int NPC=1>`
		`void extractChannel(xf::Mat<SRC_T, ROWS, COLS, NPC> & _src_mat, xf::Mat<DST_T, ROWS, COLS, NPC> & _dst_mat, uint16_t _channel)`

Threshold	`template<int ROWS, int COLS, int SRC_T, int DST_T>`
	`void Threshold(`
	`Mat<ROWS, COLS, SRC_T>& src,`
	`Mat<ROWS, COLS, DST_T>& dst,`
	`HLS_TNAME(SRC_T) thresh,`
	`HLS_TNAME(DST_T) maxval,`
	`int thresh_type)`
		`template<int THRESHOLD_TYPE, int SRC_T, int ROWS, int COLS,int NPC=1>`
		`void Threshold(xf::Mat<SRC_T, ROWS, COLS, NPC> & _src_mat,xf::Mat<SRC_T, ROWS, COLS, NPC> & _dst_mat,short int thresh,short int maxval )`

Scale	`template<int ROWS, int COLS, int SRC_T, int DST_T, typename P_T>` `void Scale(Mat<ROWS, COLS, SRC_T>& src,Mat<ROWS, COLS, DST_T>& dst, P_T scale=1.0,P_T shift=0.0)`	`template< int SRC_T,int DST_T, int ROWS, int COLS, int NPC = 1>` `void scale(xf::Mat<SRC_T, ROWS, COLS, NPC> & src1, xf::Mat<DST_T, ROWS, COLS, NPC> & dst,float scale, float shift)`

InitUndistortRectifyMapInverse	`template<typename CMT, typename DT, typename ICMT, int ROWS, int COLS, int MAP1_T, int MAP2_T, int N>`
	`void InitUndistortRectifyMapInverse (`
	`Window<3,3, CMT> cameraMatrix,DT(&distCoeffs)[N],Window<3,3, ICMT> ir, Mat<ROWS, COLS, MAP1_T>  &map1,Mat<ROWS, COLS, MAP2_T>  &map2,int noRotation=false)`
		`template< int CM_SIZE, int DC_SIZE, int MAP_T, int ROWS, int COLS, int NPC >`
		`void InitUndistortRectifyMapInverse (`
		`ap_fixed<32,12> *cameraMatrix,`
		`ap_fixed<32,12> *distCoeffs,`
		`ap_fixed<32,12> *ir,`
		`xf::Mat<MAP_T, ROWS, COLS, NPC> &_mapx_mat,xf::Mat<MAP_T, ROWS, COLS, NPC> &_mapy_mat,int _cm_size, int _dc_size)`

Avg, mean, AvgStddev	`template<typename DST_T, int ROWS, int COLS, int SRC_T>` `DST_T Mean(Mat<ROWS, COLS, SRC_T>& src)`	`template<int SRC_T,int ROWS, int COLS,int NPC=1>void meanStdDev(xf::Mat<SRC_T, ROWS, COLS, NPC> & _src,unsigned short* _mean,unsigned short* _stddev)`

Table 9: **Video Processing Functions** *(cont'd)*

Functions	HLS Video Library -API	xfOpenCV Library-API
CvtColor	`template<typename CONVERSION,int SRC_T, int DST_T,int ROWS,int COLS> void CvtColor(Mat<ROWS, COLS, SRC_T> &_src, Mat<ROWS, COLS, DST_T> &_dst)`	Color Conversion

Note: All the functions except Reduce can process N-pixels per clock where N is power of 2.

Using the xfOpenCV Library

This section describes using the xfOpenCV library in the SDx development environment.

Note: The instructions in this section assume that you have downloaded and installed all the required packages. For more information, see the Prerequisites.

`include` folder constitutes all the necessary components to build a Computer Vision or Image Processing pipeline using the library. The folders `common` and `core` contain the infrastructure that the library functions need for basic functions, Mat class, and macros. The library functions are categorized into three folders, `features`, `video` and `imgproc` based on the operation they perform. The names of the folders are self-explanatory.

To work with the library functions, you need to include the path to the The xfOpenCV library is structured as shown in the following table. The `include` folder in the SDx project. You can include relevant header files for the library functions you will be working with after you source the `include` folder's path to the compiler. For example, if you would like to work with Harris Corner Detector and Bilateral Filter, you must use the following lines in the host code:

```
#include “features/xf_harris.hpp” //for Harris Corner Detector
#include “imgproc/xf_bilateral_filter.hpp” //for Bilateral Filter
#include “video/xf_kalmanfilter.hpp”
```

After the headers are included, you can work with the library functions as described in the Chapter 5: xfOpenCV Library API Reference using the examples in the `examples` folder as reference.

The following table gives the name of the header file, including the folder name, which contains the library function.

Table 10: **xfOpenCV Library Contents**

Function Name	File Path in the `include` folder
xf::accumulate	imgproc/xf_accumulate_image.hpp
xf::accumulateSquare	imgproc/xf_accumulate_squared.hpp
xf::accumulateWeighted	imgproc/xf_accumulate_weighted.hpp
The xfOpenCV library is structured as shown in the following table.xf::absdiff, xf::add, xf::subtract, xf::bitwise_and, xf::bitwise_or, xf::bitwise_not, xf::bitwise_xor,xf::multiply ,xf::Max, xf::Min, xf::compare, xf::zero, xf::addS, xf::SubS, xf::SubRS ,xf::compareS, xf::MaxS, xf::MinS, xf::set	core/xf_arithm.hpp
xf::addWeighted	imgproc/xf_add_weighted.hpp
xf::bilateralFilter	imgproc/xf_histogram.hpp
xf::boxFilter	imgproc/xf_box_filter.hpp
xf::boundingbox	imgproc/xf_boundingbox.hpp
xf::Canny	imgproc/xf_canny.hpp
xf::Colordetect	imgproc/xf_colorthresholding.hpp, imgproc/xf_bgr2hsv.hpp, imgproc/xf_erosion.hpp, imgproc/xf_dilation.hpp
xf::merge	imgproc/xf_channel_combine.hpp
xf::extractChannel	imgproc/xf_channel_extract.hpp
xf::convertTo	imgproc/xf_convert_bitdepth.hpp
xf::crop	imgproc/xf_crop.hpp
xf::filter2D	imgproc/xf_custom_convolution.hpp
xf::nv122iyuv, xf::nv122rgba, xf::nv122yuv4, xf::nv212iyuv, xf::nv212rgba, xf::nv212yuv4, xf::rgba2yuv4, xf::rgba2iyuv, xf::rgba2nv12, xf::rgba2nv21, xf::uyvy2iyuv, xf::uyvy2nv12, xf::uyvy2rgba, xf::yuyv2iyuv, xf::yuyv2nv12, xf::yuyv2rgba,xf::rgb2iyuv,xf::rgb2nv12,xf::rgb2nv21,xf::rgb2yuv4,xf::rgb2uyvy,xf::rgb2yuyv,xf::rgb2bgr,xf::bgr2uyvy,xf::bgr2yuyv,xf::bgr2rgb,xf::bgr2nv12,xf::bgr2nv21,xf::iyuv2nv12,xf::iyuv2rgba,xf::iyuv2rgb,xf::iyuv2yuv4,xf::nv122uyvy,xf::nv122yuyv,xf::nv122nv21,xf::nv212rgb,xf::nv212bgr,xf::nv212uyvy,xf::nv212yuyv,xf::nv212nv12,xf::uyvy2rgb,xf::uyvy2bgr,xf::uyvy2yuyv,xf::yuyv2rgb,xf::yuyv2bgr,xf::yuyv2uyvy,xf::rgb2gray,xf::bgr2gray,xf::gray2rgb,xf::gray2bgr,xf::rgb2xyz,xf::bgr2xyz...	imgproc/xf_cvt_color.hpp
xf::dilate	imgproc/xf_dilation.hpp
xf::demosaicing	imgproc/xf_demosaicing.hpp
xf::erode	imgproc/xf_erosion.hpp
xf::fast	features/xf_fast.hpp
xf::GaussianBlur	imgproc/xf_gaussian_filter.hpp
xf::cornerHarris	features/xf_harris.hpp
xf::calcHist	imgproc/xf_histogram.hpp
xf::equalizeHist	imgproc/xf_hist_equalize.hpp
xf::HOGDescriptor	imgproc/xf_hog_descriptor.hpp
xf::Houghlines	imgproc/xf_houghlines.hpp
xf::inRange	imgproc/xf_inrange.hpp

Table 10: **xfOpenCV Library Contents** *(cont'd)*

Function Name	File Path in the `include` folder
xf::integralImage	imgproc/xf_integral_image.hpp
xf::densePyrOpticalFlow	video/xf_pyr_dense_optical_flow.hpp
xf::DenseNonPyrLKOpticalFlow	video/xf_dense_npyr_optical_flow.hpp
xf::LUT	imgproc/xf_lut.hpp
xf::KalmanFilter	video/xf_kalmanfilter.hpp
xf::magnitude	core/xf_magnitude.hpp
xf::MeanShift	imgproc/xf_mean_shift.hpp
xf::meanStdDev	core/xf_mean_stddev.hpp
xf::medianBlur	imgproc/xf_median_blur.hpp
xf::minMaxLoc	core/xf_min_max_loc.hpp
xf::OtsuThreshold	imgproc/xf_otsuthreshold.hpp
xf::phase	core/xf_phase.hpp
xf::paintmask	imgproc/xf_paintmask.hpp
xf::pyrDown	imgproc/xf_pyr_down.hpp
xf::pyrUp	imgproc/xf_pyr_up.hpp
xf::reduce	imgrpoc/xf_reduce.hpp
xf::remap	imgproc/xf_remap.hpp
xf::resize	imgproc/xf_resize.hpp
xf::scale	imgproc/xf_scale.hpp
xf::Scharr	imgproc/xf_scharr.hpp
xf::SemiGlobalBM	imgproc/xf_sgbm.hpp
xf::Sobel	imgproc/xf_sobel.hpp
xf::StereoPipeline	imgproc/xf_stereo_pipeline.hpp
xf::sum	imgproc/xf_sum.hpp
xf::StereoBM	imgproc/xf_stereoBM.hpp
xf::SVM	imgproc/xf_svm.hpp
xf::Threshold	imgproc/xf_threshold.hpp
xf::warpTransform	imgproc/xf_warp_transform.hpp

The different ways to use the xfOpenCV library examples are listed below:

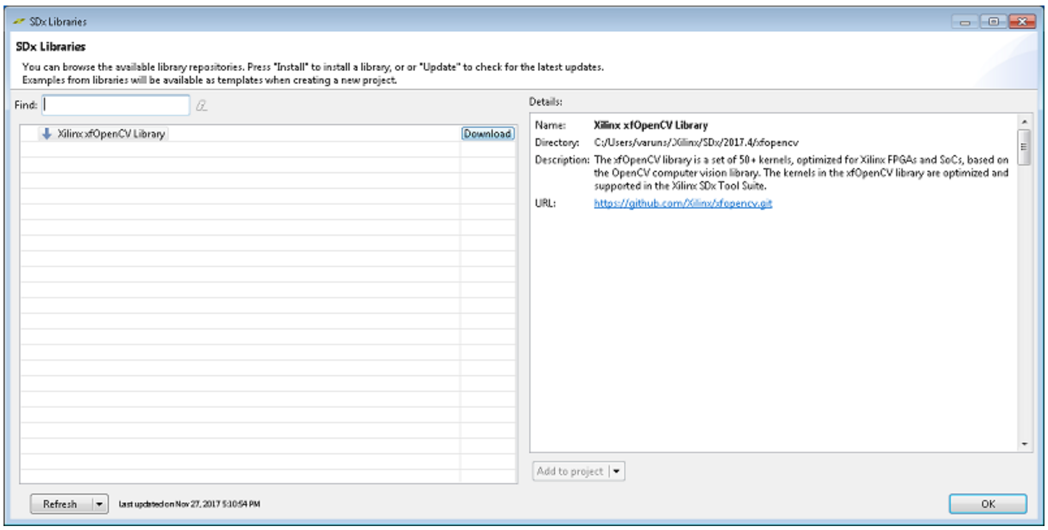
- Downloading and Using xfOpenCV Libraries from SDx GUI
- Building a Project Using the Example Makefiles on Linux
- Using reVISION Samples on the reVISION Platform
- Using the xfOpenCV Library on a non-reVISION Platform

Downloading and Using xfOpenCV Libraries from SDx GUI

You can download xfOpenCV directly from SDx GUI. To build a project using the example makefiles on the Linux platform:

1. From SDx IDE, click **Xilinx** and select **SDx Libraries**.
2. Click **Download** next to the **Xilinx xfOpenCV Library**.

Figure 2: **SDx Libraries**



The library is downloaded into `<home directory>/Xilinx/SDx/2019.1/xfopencv`. After the library is downloaded, the entire set of examples in the library are available in the list of templates while creating a new project.

Note: The library can be added to any project from the IDE menu options.

3. To add a library to a project, from SDx IDE, click **Xilinx** and select **SDx Libraries**.
4. Select **Xilinx xfOpenCV Library** and click **Add to project**. The dropdown menu consists of options of which project the libraries need to be included to.

All the headers as part of the `include/` folder in xfOpenCV library would be copied into the local project directory as `<project_dir>/libs/xfopencv/include`. All the settings required for the libraries to be run are also set when this action is completed.

Building a Project Using the Example Makefiles on Linux

Use the following steps to build a project using the example makefiles on the Linux platform:

1. Open a terminal.
2. When building for revision platform, set the environment variable SYSROOT to *<the path to platform folder>/sw/a53_linux/a53_linux/sysroot/aarch64-xilinx-linux.*
3. Change the platform variable to point to the downloaded platform folder in makefile. Ensure that the folder name of the downloaded platform is unchanged.
4. When building for revision platform , change IDIRS and LDIRS variables in the Makefile as follows:

```
IDIRS = -I. -I${SYSROOT}/usr/include -I ../../include
LDIRS = --sysroot=${SYSROOT} -L=/lib -L=/usr/lib -Wl,-rpath-link=$
{SYSROOT}/lib,-rpath-link=${SYSROOT}/usr/lib
```

5. Change the directory to the location where you want to build the example.

```
cd <path to example>
```

6. When building for revision platform , add `#include"opencv2/imgcodecs/imgcodecs.hpp"` in `xf_headers.h` file ,both in if and else part.
7. Set the environment variables to run SDx development environment.
   - For c shell:

   ```
   source <SDx tools install path>/settings.csh
   ```

   - For bash shell:

   ```
   source <SDx tools install path>/settings.sh
   ```

8. Type the `make` command in the terminal. The `sd_card` folder is created and can be found in the `<path to example>` folder.

Note: Ignore 2,4 and 6 steps when building for Non revision platforms.

Using reVISION Samples on the reVISION Platform

Use the following steps to run a unit test for bilateral filter on zcu104_rv_ss:

1. Launch the SDx development environment using the desktop icon or the **Start** menu.

   The **Workspace Launcher** dialog appears.

2. Click **Browse** to enter a workspace folder used to store your projects (you can use workspace folders to organize your work), then click **OK** to dismiss the **Workspace Launcher** dialog.

   Note: Before launching the SDx IDE on Linux, ensure that you use the same shell that you have used to set the `$SYSROOT` environment variable. This is usually the file path to the Linux root file system.

   The SDx development environment window opens with the **Welcome** tab visible when you create a new workspace. The **Welcome** tab can be closed by clicking the **X** icon or minimized if you do not wish to use it.

3. Select **File → New → Xilinx SDx Project** from the SDx development environment menu bar.

   The **New Project** dialog box opens.

4. Specify the name of the project. For example **Bilateral**.
5. Click **Next**.

   The the **Choose Hardware Platform** page appears.

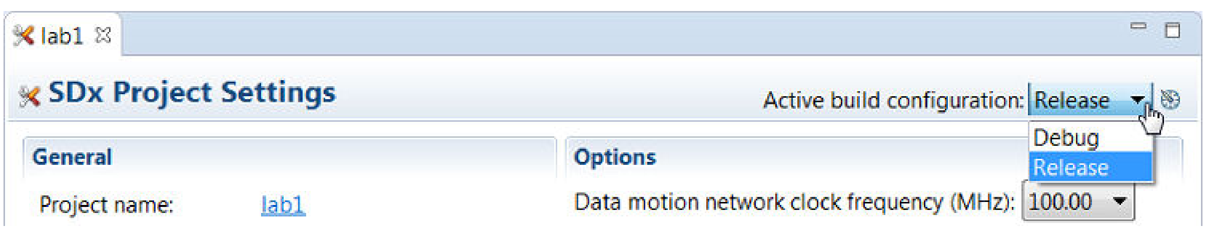
6. From the **Choose Hardware Platform** page, click the **Add Custom Platform** button.
7. Browse to the directory where you extracted the reVISION platform files. Ensure that you select the `zcu104_rv_ss` folder.
8. From the **Choose Hardware Platform** page, select **zcu104_rv_ss (custom)**.
9. Click **Next**.

   The **Templates** page appears, containing source code examples for the selected platform.

10. From the list of application templates, select **bilateral - File I/O** and click **Finish**.
11. Click the **Active build configurations** drop-down from the **SDx Project Settings** window, to select the active configuration or create a build configuration.

    The standard build configurations are Debug and Release. To get the best runtime performance, switch to use the **Release** build configuration as it uses a higher compiler optimization setting than the Debug build configuration.

Figure 3: **SDx Project Settings - Active Build Configuration**



12. Set the **Data motion network clock frequency (MHz)** to the required frequency, on the **SDx Project Settings** page.
13. Right-click the project and select **Build Project** or press Ctrl+B keys to build the project, in the **Project Explorer** view.
14. Copy the contents of the newly created `sd_card` folder to the SD card. The `sd_card` folder contains all the files required to run designs on the `ZCU104` board.
15. Insert the SD card in the `ZCU104` board card slot and switch it ON.

    Note: A serial port emulator (Teraterm/ minicom) is required to interface the user commands to the board.

16. Upon successful boot, run the following command in the Teraterm terminal (serial port emulator.)

```
#cd /media/card
#remount
```

17. Run the `.elf` file for the respective functions.

    For more information, see the Using the xfOpenCV Library Functions on Hardware.

Using the xfOpenCV Library on a non-reVISION Platform

This section describes using the xfOpenCV library on a non-reVISION platform, in the SDx™ development environment. The examples in xfOpenCV require OpenCV libraries for successful compilation. As non-reVISION platform may or may not contain opencv libs, as a perquisites it is required to install/compile opencv libraries(with compatible libjpeg.so).

Note: The instructions in this section assume that you have downloaded and installed all the required packages. For more information, see the Prerequisites.

Use the following steps to import the xfOpenCV library into a SDx project and execute it on a custom platform:

1. Launch the SDx development environment using the desktop icon or the **Start** menu.

   The **Workspace Launcher** dialog appears.

2. Click **Browse** to enter a workspace folder used to store your projects (you can use workspace folders to organize your work), then click **OK** to dismiss the **Workspace Launcher** dialog.

   The SDx development environment window opens with the **Welcome** tab visible when you create a new workspace. The **Welcome** tab can be closed by clicking the **X** icon or minimized if you do not wish to use it.

3. Select **File→New→Xilinx SDx Project** from the SDx development environment menu bar.

   The **New Project** dialog box opens.

4. Specify the name of the project. For example **Test**.
5. Click **Next**.

   The the **Choose Hardware Platform** page appears.

6. From the **Choose Hardware Platform** page, select a suitable platform. For example, **zcu102**.
7. Click **Next**.

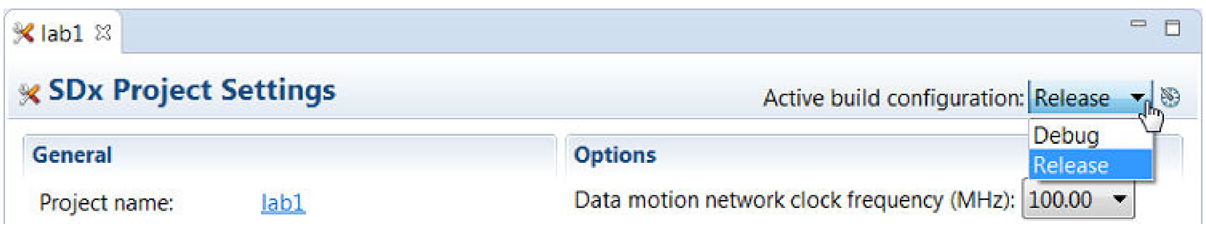
   The **Choose Software Platform and Target CPU** page appears.

8. From the **Choose Software Platform and Target CPU** page, select an appropriate software platform and the target CPU. For example, select **A9** from the **CPU** dropdown list for ZC702 and ZC706 reVISION platforms.
9. Click **Next**. The **Templates** page appears, containing source code examples for the selected platform.
10. From the list of application templates, select **Empty Application** and click **Finish**.

    The **New Project** dialog box closes. A new project with the specified configuration is created. The **SDx Project Settings** view appears. Notice the progress bar in the lower right border of the view, Wait for a few moments for the **C/C++ Indexer** to finish.

11. The standard build configurations are Debug and Release. To get the best run-time performance, switch to use the **Release** build configuration as it uses a higher compiler optimization setting than the Debug build configuration.

Figure 4: **SDx Project Settings - Active Build Configuration**



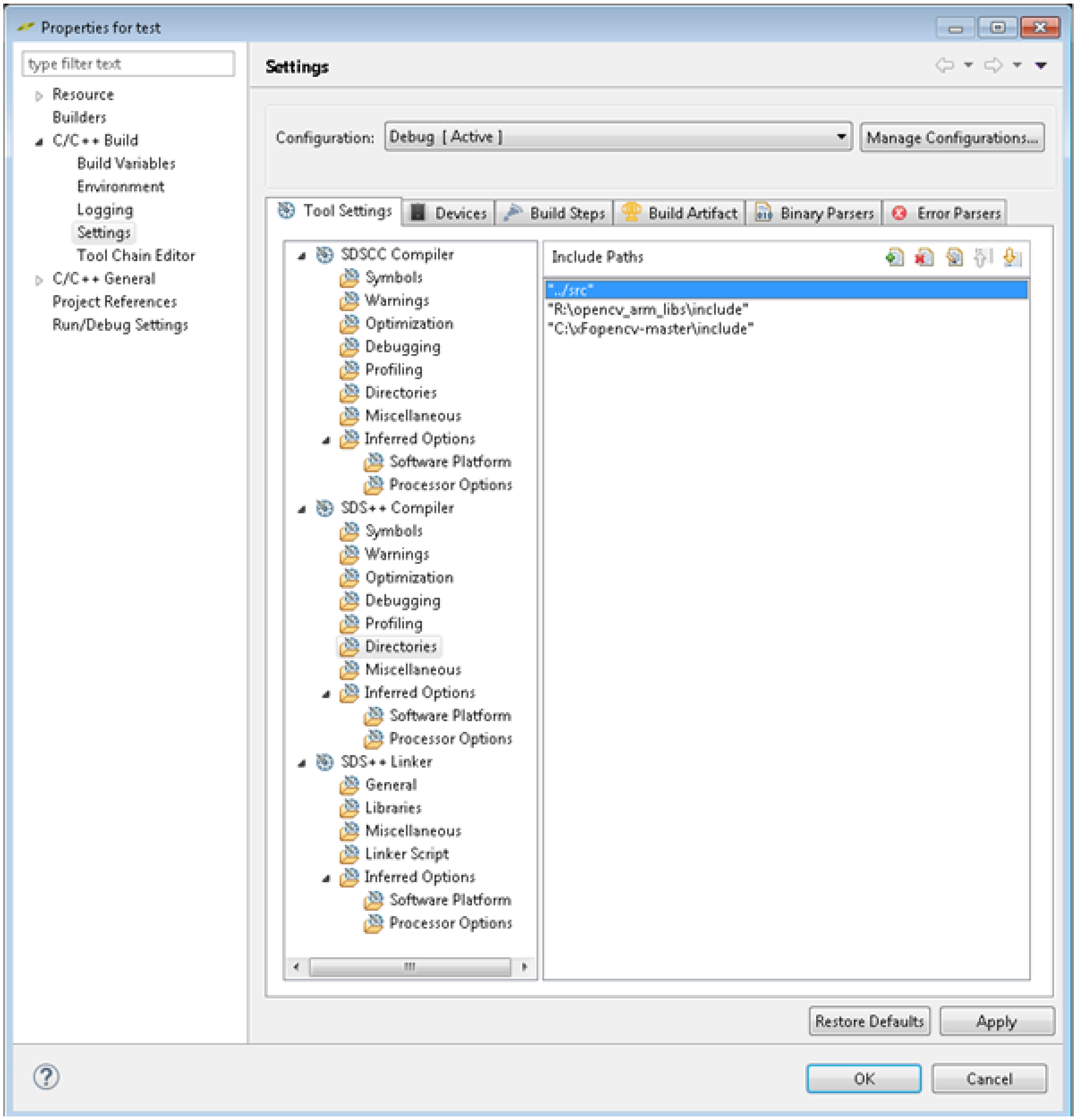
12. Set the **Data motion network clock frequency (MHz)** to the required frequency, on the **SDx Project Settings** page.
13. Select the **Generate bitstream** and **Generate SD card image** check boxes.
14. Right-click on the newly created project in the **Project Explorer** view.
15. From the context menu that appears, select **C/C++ Build Settings**.

    The **Properties for <project>** dialog box appears.

16. Click the **Tool Settings** tab.
17. Expand the **SDS++ Compiler → Directories** tree.
18. Click the  icon to add the `"<xfopencv_location>\include"` and `"<OpenCV_location>\include"` folder locations to the **Include Paths** list.

    ***Note*:** The OpenCV library is not provided by Xilinx for custom platforms. You are required to provide the library. Use the reVISION platform in order to use the OpenCV library provided by Xilinx.

Figure 5: **SDS++ Compiler Settings**



19. In the same page, under **SDS++ Compiler → Inferred Options → Software Platform**, specify "-hls-target 1" in the Software Platform Inferred Flags.

20. Click **Apply**.

21. Expand the **SDS++ Linker → Libraries** tree.

22. Click the  icon and add the following libraries to the **Libraries(-l)** list. These libraries are required by OpenCV.

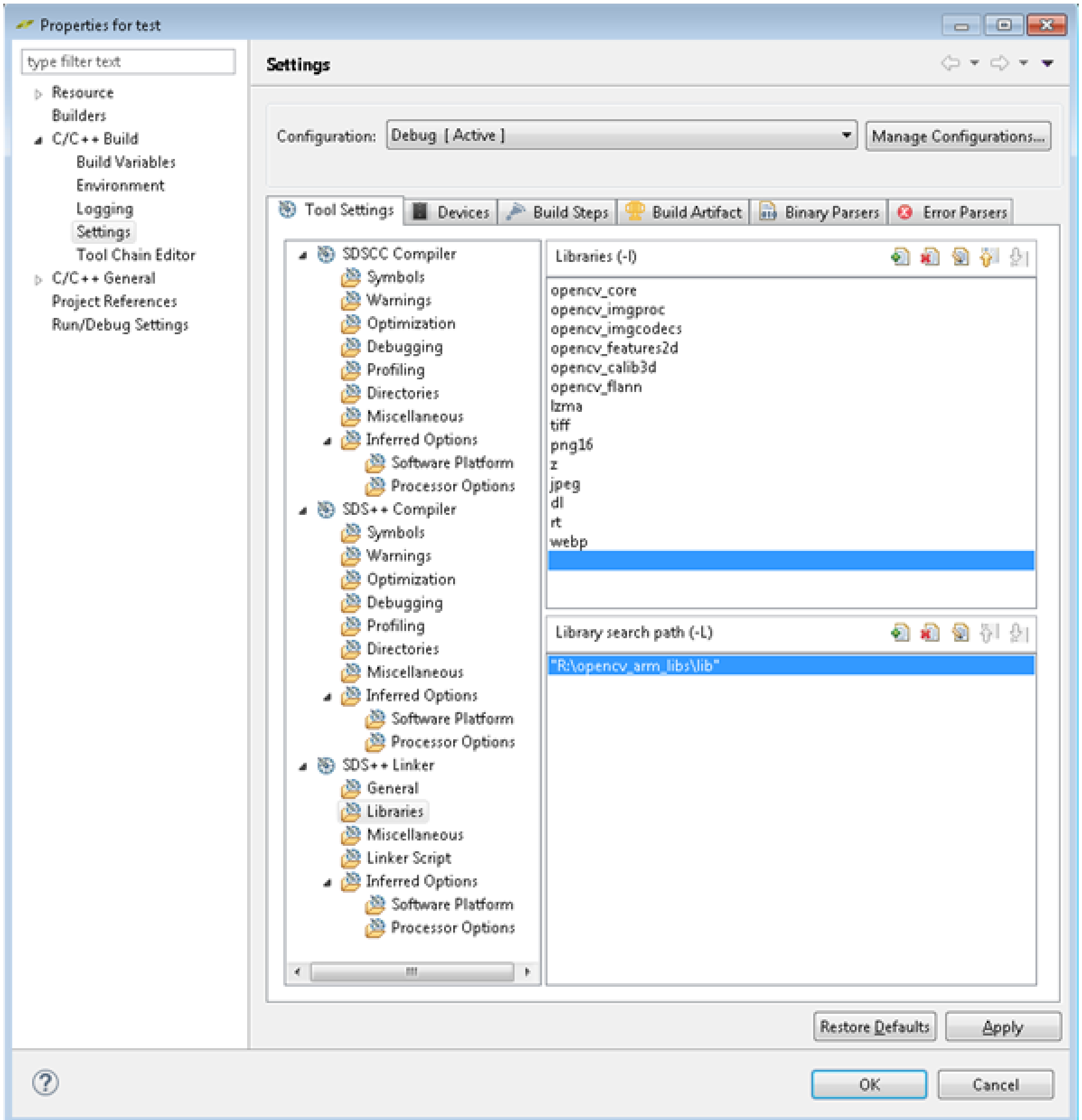
- opencv_core

- opencv_imgproc
- opencv_imgcodecs
- opencv_features2d
- opencv_calib3d
- opencv_flann
- opencv_video
- opencv_videoio

23. Click the icon and add <opencv_Location>/lib folder location to the **Libraries search path (-L)** list.

    ***Note*:** The OpenCV library is not provided by Xilinx for custom platforms. You are required to provide the library. Use the reVISION platform in order to use the OpenCV library provided by Xilinx.

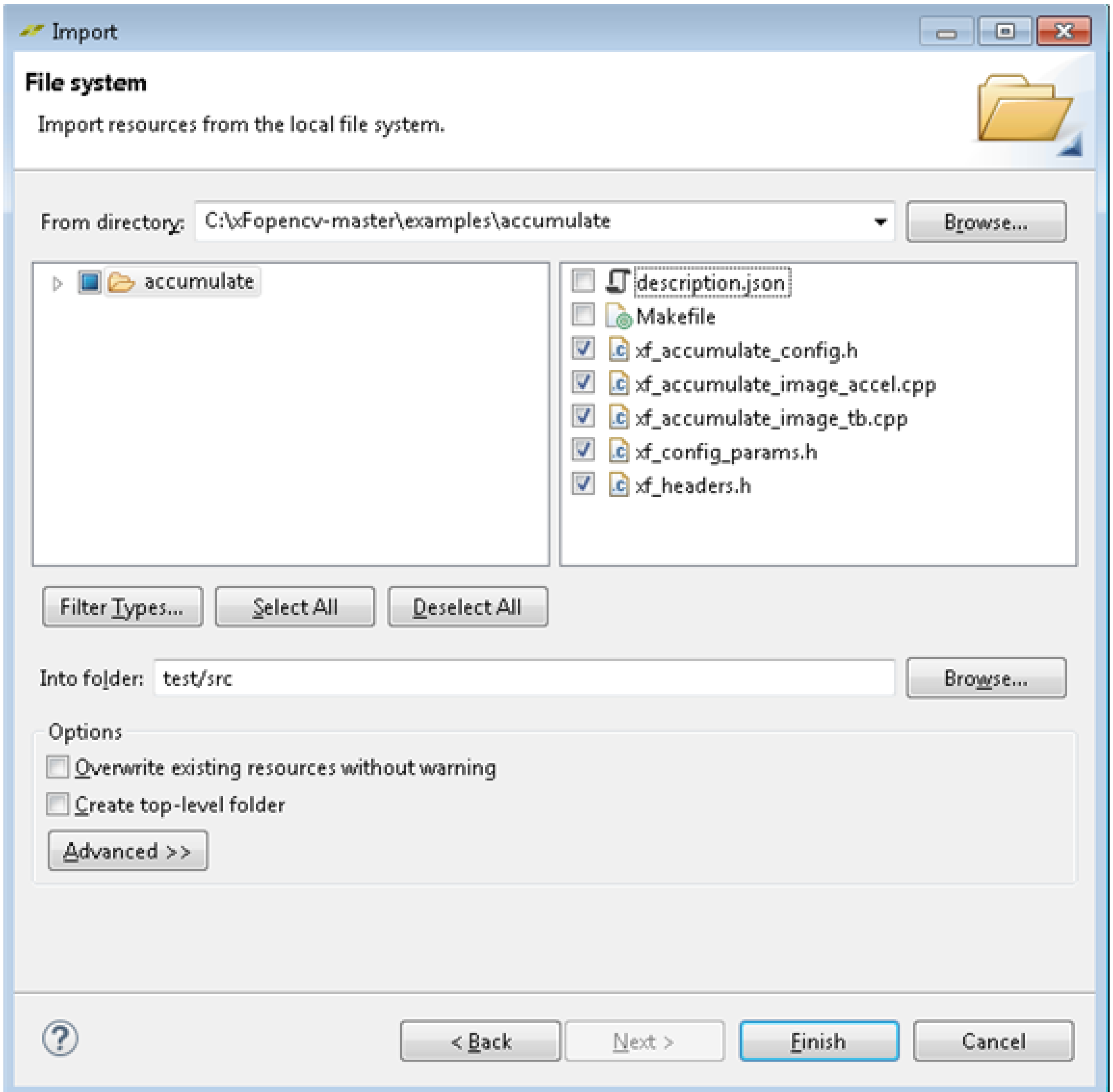
Figure 6: **SDS++ Linker Settings**



24. Click **Apply** to save the configuration.
25. Click **OK** to close the **Properties for <project>** dialog box.
26. Expand the newly created project tree in the **Project Explorer** view.
27. Right-click the **src** folder and select **Import**. The **Import** dialog box appears.
28. Select **File System** and click **Next**.
29. Click **Browse** to navigate to the <xfopencv_Location>/examples folder location.

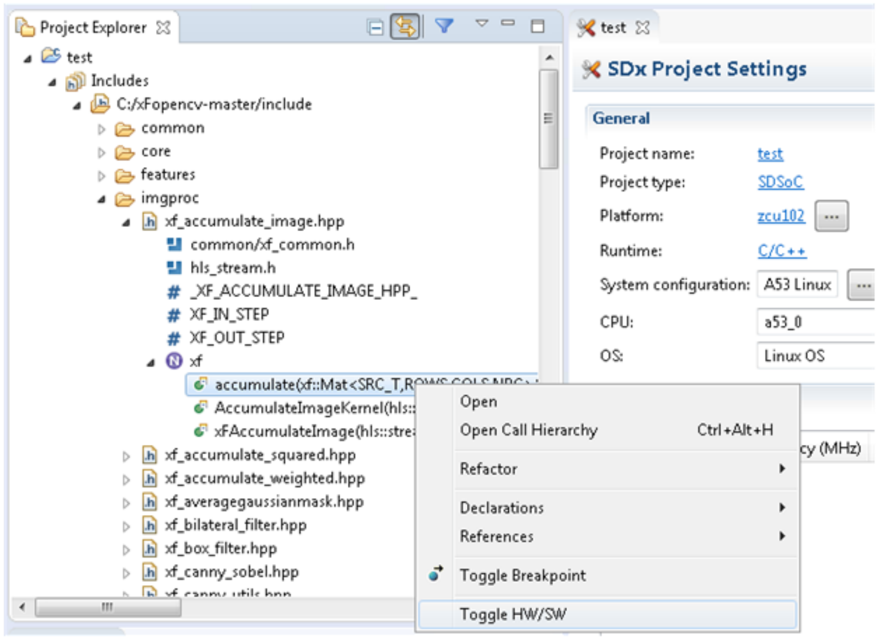
30. Select the folder that corresponds to the library that you desire to import. For example, `accumulate`.

Figure 7: **Import Library Example Source Files**



31. Right-click the library function in the **Project Explorer** view and select **Toggle HW/SW** to move the function to the hardware.

Figure 8: **Moving a Library Function to the Hardware**



32. Right-click the project and select **Build Project** or press Ctrl+B keys to build the project, in the **Project Explorer** view.

    The build process may take anytime between few minutes to several hours, depending on the power of the host machine and the complexity of the design. By far, the most time is spent processing the routines that have been tagged for realization in hardware.

33. Copy the contents of the newly created `.\<workspace>\<function>\Release\sd_card` folder to the SD card. The `sd_card` folder contains all the files required to run designs on a board.

34. Insert the SD card in the board card slot and switch it ON.

    ***Note*:** A serial port emulator (Teraterm/ minicom) is required to interface the user commands to the board.

35. Upon successful boot, navigate to the `./mnt` folder and run the following command at the prompt:

```
#cd /mnt
```

    ***Note*:** It is assumed that the OpenCV libraries are a port of the root filesystem. If not, add the location of OpenCV libraries to `LD_LIBRARY_PATH` using the `$ export LD_LIBRARY_PATH=<location of OpenCV libraries>/lib` command.

36. Run the `.elf` executable file. For more information, see the Using the xfOpenCV Library Functions on Hardware.

Changing the Hardware Kernel Configuration

Use the following steps to change the hardware kernel configuration:

1. Update the `<path to xfOpenCV git folder>/xfOpenCV/examples/<function>/xf_config_params.h` file.
2. Update the makefile along with the `xf_config_params.h` file:
    a. Find the line with the function name in the makefile. For bilateral filter, the line in the makefile will be `xf::BilateralFilter<3,1,0,1080,1920,1>`.
    b. Update the template parameters in the makefile to reflect changes made in the `xf_config_params.h` file. For more details, see the Chapter 5: xfOpenCV Library API Reference.

Using the xfOpenCV Library Functions on Hardware

The following table lists the xfOpenCV library functions and the command to run the respective examples on hardware. It is assumed that your design is completely built and the board has booted up correctly.

Table 11: **Using the xfOpenCV Library Function on Hardware**

Example	Function Name	Usage on Hardware
accumulate	xf::accumulate	./<executable name>.elf <path to input image 1> <path to input image 2>
accumulatesquared	xf::accumulateSquare	./<executable name>.elf <path to input image 1> <path to input image 2>
accumulateweighted	xf::accumulateWeighted	./<executable name>.elf <path to input image 1> <path to input image 2>
addS	xf::addS	./<executable name>.elf <path to input image>
arithm	xf::absdiff, xf::add, xf::subtract, xf::bitwise_and, xf::bitwise_or, xf::bitwise_not, xf::bitwise_xor	./<executable name>.elf <path to input image 1> <path to input image 2>
addweighted	xf::addWeighted	./<executable name>.elf <path to input image 1> <path to input image 2>
Bilateralfilter	xf::bilateralFilter	./<executable name>.elf <path to input image>
Boxfilter	xf::boxFilter	./<executable name>.elf <path to input image>
Boundingbox	xf::boundingbox	./<executable name>.elf <path to input image> <No of ROI's>

Table 11: **Using the xfOpenCV Library Function on Hardware** *(cont'd)*

Example	Function Name	Usage on Hardware
Canny	xf::Canny	./<executable name>.elf <path to input image>
channelcombine	xf::merge	./<executable name>.elf <path to input image 1> <path to input image 2> <path to input image 3> <path to input image 4>
Channelextract	xf::extractChannel	./<executable name>.elf <path to input image>
Colordetect	xf::bgr2hsv, xf::colorthresholding, xf:: erode, and xf:: dilate	./<executable name>.elf <path to input image>
compare	xf::compare	./<executable name>.elf <path to input image 1> <path to input image 2>
compareS	xf::compareS	./<executable name>.elf <path to input image>
Convertbitdepth	xf::convertTo	./<executable name>.elf <path to input image>
Cornertracker	xf::cornerTracker	./exe <input video> <no. of frames> <Harris Threshold> <No. of frames after which Harris Corners are Reset>
crop	xf::crop	./<executable name>.elf <path to input image>
Customconv	xf::filter2D	./<executable name>.elf <path to input image>
cvtcolor IYUV2NV12	xf::iyuv2nv12	./<executable name>.elf <path to input image 1> <path to input image 2> <path to input image 3>
cvtcolor IYUV2RGBA	xf::iyuv2rgba	./<executable name>.elf <path to input image 1> <path to input image 2> <path to input image 3>
cvtcolor IYUV2YUV4	xf::iyuv2yuv4	./<executable name>.elf <path to input image 1> <path to input image 2> <path to input image 3> <path to input image 4> <path to input image 5> <path to input image 6>
cvtcolor NV122IYUV	xf::nv122iyuv	./<executable name>.elf <path to input image 1> <path to input image 2>
cvtcolor NV122RGBA	xf::nv122rgba	./<executable name>.elf <path to input image 1> <path to input image 2>
cvtcolor NV122YUV4	xf::nv122yuv4	./<executable name>.elf <path to input image 1> <path to input image 2>
cvtcolor NV212IYUV	xf::nv212iyuv	./<executable name>.elf <path to input image 1> <path to input image 2>
cvtcolor NV212RGBA	xf::nv212rgba	./<executable name>.elf <path to input image 1> <path to input image 2>
cvtcolor NV212YUV4	xf::nv212yuv4	./<executable name>.elf <path to input image 1> <path to input image 2>
cvtcolor RGBA2YUV4	xf::rgba2yuv4	./<executable name>.elf <path to input image>
cvtcolor RGBA2IYUV	xf::rgba2iyuv	./<executable name>.elf <path to input image>

Table 11: **Using the xfOpenCV Library Function on Hardware** *(cont'd)*

Example	Function Name	Usage on Hardware
cvtcolor RGBA2NV12	xf::rgba2nv12	./<executable name>.elf <path to input image>
cvtcolor RGBA2NV21	xf::rgba2nv21	./<executable name>.elf <path to input image>
cvtcolor UYVY2IYUV	xf::uyvy2iyuv	./<executable name>.elf <path to input image>
cvtcolor UYVY2NV12	xf::uyvy2nv12	./<executable name>.elf <path to input image>
cvtcolor UYVY2RGBA	xf::uyvy2rgba	./<executable name>.elf <path to input image>
cvtcolor YUYV2IYUV	xf::yuyv2iyuv	./<executable name>.elf <path to input image>
cvtcolor YUYV2NV12	xf::yuyv2nv12	./<executable name>.elf <path to input image>
cvtcolor YUYV2RGBA	xf::yuyv2rgba	./<executable name>.elf <path to input image>
Demosaicing	xf::demosaicing	./<executable name>.elf <path to input image>
Difference of Gaussian	xf:: GaussianBlur, xf:: duplicateMat, xf:: delayMat, and xf::subtract	./<exe-name>.elf <path to input image>
Dilation	xf::dilate	./<executable name>.elf <path to input image>
Erosion	xf::erode	./<executable name>.elf <path to input image>
Fast	xf::fast	./<executable name>.elf <path to input image>
Gaussianfilter	xf::GaussianBlur	./<executable name>.elf <path to input image>
Harris	xf::cornerHarris	./<executable name>.elf <path to input image>
Histogram	xf::calcHist	./<executable name>.elf <path to input image>
Histequialize	xf::equalizeHist	./<executable name>.elf <path to input image>
Hog	xf::HOGDescriptor	./<executable name>.elf <path to input image>
Houghlines	xf::HoughLines	./<executable name>.elf <path to input image>
inRange	xf::inRange	./<executable name>.elf <path to input image>
Integralimg	xf::integralImage	./<executable name>.elf <path to input image>
Lkdensepyrof	xf::densePyrOpticalFlow	./<executable name>.elf <path to input image 1> <path to input image 2>
Lknpyroflow	xf::DenseNonPyrLKOpticalFlow	./<executable name>.elf <path to input image 1> <path to input image 2>
Lut	xf::LUT	./<executable name>.elf <path to input image>

Table 11: **Using the xfOpenCV Library Function on Hardware** *(cont'd)*

Example	Function Name	Usage on Hardware
Kalman Filter	xf::KalmanFilter	./<executable name>.elf
Magnitude	xf::magnitude	./<executable name>.elf <path to input image>
Max	xf::Max	./<executable name>.elf <path to input image 1> <path to input image 2>
MaxS	xf::MaxS	./<executable name>.elf <path to input image>
meanshifttracking	xf::MeanShift	./<executable name>.elf <path to input video/input image files> <Number of objects to track>
meanstddev	xf::meanStdDev	./<executable name>.elf <path to input image>
medianblur	xf::medianBlur	./<executable name>.elf <path to input image>
Min	xf::Min	./<executable name>.elf <path to input image 1> <path to input image 2>
MinS	xf::MinS	./<executable name>.elf <path to input image>
Minmaxloc	xf::minMaxLoc	./<executable name>.elf <path to input image>
otsuthreshold	xf::OtsuThreshold	./<executable name>.elf <path to input image>
paintmask	xf::paintmask	./<executable name>.elf <path to input image>
Phase	xf::phase	./<executable name>.elf <path to input image>
Pyrdown	xf::pyrDown	./<executable name>.elf <path to input image>
Pyrup	xf::pyrUp	./<executable name>.elf <path to input image>
reduce	xf::reduce	./<executable name>.elf <path to input image>
remap	xf::remap	./<executable name>.elf <path to input image> <path to mapx data> <path to mapy data>
Resize	xf::resize	./<executable name>.elf <path to input image>
scale	xf::scale	./<executable name>.elf <path to input image>
scharrfilter	xf::Scharr	./<executable name>.elf <path to input image>
set	xf::set	./<executable name>.elf <path to input image>
SemiGlobalBM	xf::SemiGlobalBM	./<executable name>.elf <path to left image> <path to right image>
sobelfilter	xf::Sobel	./<executable name>.elf <path to input image>
stereopipeline	xf::StereoPipeline	./<executable name>.elf <path to left image> <path to right image>

Table 11: **Using the xfOpenCV Library Function on Hardware** *(cont'd)*

Example	Function Name	Usage on Hardware
stereolbm	xf::StereoBM	./<executable name>.elf <path to left image> <path to right image>
subRS	xf::SubRS	./<executable name>.elf <path to input image>
subS	xf::SubS	./<executable name>.elf <path to input image>
sum	xf::sum	./<executable name>.elf <path to input image 1> <path to input image 2>
Svm	xf::SVM	./<executable name>.elf
threshold	xf::Threshold	./<executable name>.elf <path to input image>
warptransform	xf::warpTransform	./<executable name>.elf <path to input image>
zero	xf::zero	./<executable name>.elf <path to input image>

Getting Started with SDAccel

This chapter provides details on using xfOpenCV in the SDAccel™ environment. The following sections would provide a description of the methodology to create a kernel, corresponding host code and a suitable makefile to compile an xfOpenCV kernel for any of the supported platforms in SDAccel. The subsequent section also explains the methodology to verify the kernel in various emulation modes and on the hardware.

Prerequisites

1. Valid installation of SDx™ 2019.1 or later version and the corresponding licenses.
2. Install the xfOpenCV libraries, if you intend to use libraries compiled differently than what is provided in SDx.
3. Install the card for which the platform is supported in SDx 2019.1 or later versions.
4. Xilinx® Runtime (XRT) must be installed. XRT provides software interface to Xilinx FPGAs.
5. libOpenCL.so must be installed if not present along with the platform.

SDAccel Design Methodology

There are three critical components in making a kernel work on a platform using SDAccel™:

1. Host code with OpenCL constructs
2. Wrappers around HLS Kernel(s)
3. Makefile to compile the kernel for emulation or running on hardware.

Host Code with OpenCL

Host code is compiled for the host machine that runs on the host and provides the data and control signals to the attached hardware with the FPGA. The host code is written using OpenCL constructs and provides capabilities for setting up, and running a kernel on the FPGA. The following functions are executed using the host code:

1. Loading the kernel binary on the FPGA – xcl::import_binary_file() loads the bitstream and programs the FPGA to enable required processing of data.
2. Setting up memory buffers for data transfer – Data needs to be sent and read from the DDR memory on the hardware. cl::Buffers are created to allocate required memory for transferring data to and from the hardware.
3. Transfer data to and from the hardware –enqueueWriteBuffer() and enqueueReadBuffer() are used to transfer the data to and from the hardware at the required time.
4. Execute kernel on the FPGA – There are functions to execute kernels on the FPGA. There can be single kernel execution or multiple kernel execution that could be asynchronous or synchronous with each other. Commonly used command is enqueueTask().
5. Profiling the performance of kernel execution – The host code in OpenCL also enables measurement of the execution time of a kernel on the FPGA. The function used in our examples for profiling is getProfilingInfo().

Wrappers around HLS Kernel(s)

All xfOpenCV kernels are provided with C++ function templates (located at <Github repo>/include) with image containers as objects of xf::Mat class. In addition, these kernels will work either in stream based (where complete image is read continuously) or memory mapped (where image data access is in blocks).

SDAccel flow (OpenCL) requires kernel interfaces to be memory pointers with width in power(s) of 2. So glue logic is required for converting memory pointers to xf::Mat class data type and vice-versa when interacting with xfOpenCV kernel(s). Wrapper(s) are build over the kernel(s) with this glue logic. Below examples will provide a methodology to handle different kernel (xfOpenCV kernels located at <Github repo>/include) types (stream and memory mapped).

Stream Based Kernels

To facilitate the conversion of pointer to xf::Mat and vice versa, two adapter functions are included as part of xfOpenCV xf::Array2xfMat() and xf::xfMat2Array().It is necessary for the xf::Mat objects to be invoked as streams using HLS pragma with a minimum depth of 2. This results in a top-level (or wrapper) function for the kernel as shown below:

```
extern “C”
{
void func_top (ap_uint *gmem_in, ap_uint *gmem_out, ...) {
xf::Mat<…> in_mat(…), out_mat(…);
#pragma HLS stream variable=in_mat.data depth=2
#pragma HLS stream variable=out_mat.data depth=2
#pragma HLS dataflow
xf::Array2xfMat<…> (gmem_in, in_mat);
xf::xfopencv-func<…> (in_mat, out_mat…);
xf::xfMat2Array<…> (gmem_out, out_mat);
}
}
```

The above illustration assumes that the data in xf::Mat is being streamed in and streamed out. You can also create a pipeline with multiple functions in pipeline instead of just one xfopencv function.

For the stream based kernels with different inputs of different sizes, multiple instances of the adapter functions are necessary. For this,

```
extern “C” {
void func_top (ap_uint *gmem_in1, ap_uint *gmem_in2, ap_uint *gmem_in3,
ap_uint *gmem_out, ...) {
xf::Mat<...,HEIGHT,WIDTH,…> in_mat1(…), out_mat(…);
xf::Mat<...,HEIGHT/4,WIDTH,…>  in_mat2(…), in_mat3(…);
#pragma HLS stream variable=in_mat1.data depth=2
#pragma HLS stream variable=in_mat2.data depth=2
#pragma HLS stream variable=in_mat3.data depth=2
#pragma HLS stream variable=out_mat.data depth=2
#pragma HLS dataflow
xf::accel_utils obj_a, obj_b;
obj_a.Array2xfMat<…,HEIGHT,WIDTH,…> (gmem_in1, in_mat1);
obj_b.Array2xfMat<…,HEIGHT/4,WIDTH,…> (gmem_in2, in_mat2);
obj_b.Array2xfMat<…,HEIGHT/4,WIDTH,…> (gmem_in3, in_mat3);
xf::xfopencv-func(in_mat1, in_mat2, int_mat3, out_mat…);
xf::xfMat2Array<…> (gmem_out, out_mat);
}
}
```

For the stream based implementations, the data must be fetched from the input AXI and must be pushed to xfMat as required by the xfcv kernels for that particular configuration. Likewise, the same operations must be performed for the output of the xfcv kernel. To perform this, two utility functions are provided, xf::Array2xfMat() and xf::xfMat2Array().

Array2xfMat

This function converts the input array to xf::Mat. The xfOpenCV kernel would require the input to be of type, xf::Mat. This function would read from the array pointer and write into xf::Mat based on the particular configuration (bit-depth, channels, pixel-parallelism) the xf::Mat was created.

```
template <int PTR_WIDTH, int MAT_T, int ROWS, int COLS, int NPC>
void Array2xfMat(ap_uint< PTR_WIDTH > *srcPtr,
xf::Mat<MAT_T,ROWS,COLS,NPC>& dstMat)
```

Table 12: **Array2xfMat Parmater Description**

Parameter	Description
PTR_WIDTH	Data width of the input pointer. The value must be power 2, starting from 8 to 512.
MAT_T	Input Mat type. Example XF_8UC1, XF_16UC1, XF_8UC3 and XF_8UC4
ROWS	Maximum height of image
COLS	Maximum width of image

Table 12: **Array2xfMat Parmater Description** *(cont'd)*

Parameter	Description
NPC	Number of pixels computed in parallel. Example XF_NPPC1, XF_NPPC8
srcPtr	Input pointer. Type of the pointer based on the PTR_WIDTH.
dstMat	Output image of type xf::Mat

xfMat2Array

This function converts the input xf::Mat to output array. The output of the xf::kernel function will be xf::Mat, and it will require to convert that to output pointer.

```
template <int PTR_WIDTH, int MAT_T, int ROWS, int COLS, int NPC>
void xfMat2Array(xf::Mat<MAT_T,ROWS,COLS,NPC>& srcMat, ap_uint< PTR_WIDTH >
*dstPtr)
```

Table 13: **xfMat2Array Parameter Description**

Parameter	Description
PTR_WIDTH	Data width of the output pointer. The value must be power 2, from 8 to 512.
MAT_T	Input Mat type. Example XF_8UC1, XF_16UC1, XF_8UC3 and XF_8UC4
ROWS	Maximum height of image
COLS	Maximum width of image
NPC	Number of pixels computed in parallel. Example XF_NPPC1, XF_NPPC8
dstPtr	Output pointer. Type of the pointer based on the PTR_WIDTH.
srcMat	Input image of type xf::Mat

Interface pointer widths

Minimum pointer widths for different configurations is shown in the following table:

Table 14: **Minimum and maximum pointer widths for different mat types**

MAT type	Parallelism	Min PTR_WIDTH	Max PTR_WIDTH
XF_8UC1	XF_NPPC1	8	512
XF_16UC1	XF_NPPC1	16	512
XF_ 8UC1	XF_NPPC8	64	512
XF_ 16UC1	XF_NPPC8	128	512
XF_ 8UC3	XF_NPPC1	32	512
XF_ 8UC3	XF_NPPC8	256	512
XF_8UC4	XF_NPPC8	256	512
XF_8UC3	XF_NPPC16	512	512

Memorymapped Kernels

In the memory map based kernels such as crop, Mean-shift tracking and bounding box, the input read will be for particular block of memory based on the requirement for the algorithm. The streaming interfaces will require the image to be read in raster scan manner, which is not the case for the memory mapped kernels. The methodology to handle this case is as follows:

```
extern “C”
{
void func_top (ap_uint *gmem_in, ap_uint *gmem_out, ...) {
xf::Mat<…> in_mat(…,gmem_in), out_mat(…,gmem_out);
xf::kernel<…> (in_mat, out_mat…);
}
}
```

The gmem pointers must be mapped to the xf::Mat objects during the object creation, and then the memory mapped kernels are called with these mats at the interface. It is necessary that the pointer size must be same as the size required for the xf::xfopencv-func, unlike the streaming method where any higher size of the pointers (till 512-bits) are allowed.

Makefile

In the current use model, only a makefile based flow is provided to build applications with xfOpenCV on SDAccel. Examples for makefile are provided in the samples section of GitHub.

Design example Using Library on SDAccel

Following is a multi-kernel example, where different kernel runs sequentially in a pipeline to form an application. This example performs Canny edge detection, where two kernels are involved, Canny and edge tracing. Canny function will take gray-scale image as input and provided the edge information in 3 states(weak edge(1),strong edge(3) and background(0)), which is being fed into edge tracing, which filters out the weak edges. The prior works in a streaming based implementation and the later in a memory mapped manner.

Host code

The following is the Host code for the canny edge detection example. The host code sets up the OpenCL platform with the FPGA of processing required data. In the case of xfOpenCV example, the data is an image. Reading and writing of images are enabled using called to functions from xfOpenCV.

```
// setting up device and platform
    std::vector<cl::Device> devices = xcl::get_xil_devices();
    cl::Device device = devices[0];
    cl::Context context(device);
    cl::CommandQueue q(context, device,CL_QUEUE_PROFILING_ENABLE);
    std::string device_name = device.getInfo<CL_DEVICE_NAME>();

    // Kernel 1: Canny
```

```
    std::string binaryFile=xcl::find_binary_file(device_name,"krnl_canny");
    cl::Program::Binaries bins = xcl::import_binary_file(binaryFile);
    devices.resize(1);
    cl::Program program(context, devices, bins);
    cl::Kernel krnl(program,"canny_accel");

    // creating necessary cl buffers for input and output
    cl::Buffer imageToDevice(context, CL_MEM_READ_ONLY,(height*width));
    cl::Buffer imageFromDevice(context, CL_MEM_WRITE_ONLY,(height*width/4));


    // Set the kernel arguments
    krnl.setArg(0, imageToDevice);
    krnl.setArg(1, imageFromDevice);
    krnl.setArg(2, height);
    krnl.setArg(3, width);
    krnl.setArg(4, low_threshold);
    krnl.setArg(5, high_threshold);

    // write the input image data from host to device memory
    q.enqueueWriteBuffer(imageToDevice, CL_TRUE, 0,
(height*(width)),img_gray.data);
    // Profiling Objects
    cl_ulong start= 0;
    cl_ulong end = 0;
    double diff_prof = 0.0f;
    cl::Event event_sp;

    // Launch the kernel
    q.enqueueTask(krnl,NULL,&event_sp);
    clWaitForEvents(1, (const cl_event*) &event_sp);

    // profiling
    event_sp.getProfilingInfo(CL_PROFILING_COMMAND_START,&start);
    event_sp.getProfilingInfo(CL_PROFILING_COMMAND_END,&end);
    diff_prof = end-start;
    std::cout<<(diff_prof/1000000)<<"ms"<<std::endl;

    // Kernel 2: edge tracing
    cl::Kernel krnl2(program,"edgetracing_accel");

    cl::Buffer imageFromDeviceedge(context, CL_MEM_WRITE_ONLY,
(height*width));

    // Set the kernel arguments
    krnl2.setArg(0, imageFromDevice);
    krnl2.setArg(1, imageFromDeviceedge);
    krnl2.setArg(2, height);
    krnl2.setArg(3, width);

    // Profiling Objects
    cl_ulong startedge= 0;
    cl_ulong endedge = 0;
    double diff_prof_edge = 0.0f;
    cl::Event event_sp_edge;

    // Launch the kernel
    q.enqueueTask(krnl2,NULL,&event_sp_edge);
    clWaitForEvents(1, (const cl_event*) &event_sp_edge);

    // profiling
    event_sp_edge.getProfilingInfo(CL_PROFILING_COMMAND_START,&startedge);
    event_sp_edge.getProfilingInfo(CL_PROFILING_COMMAND_END,&endedge);
```

```
    diff_prof_edge = endedge-startedge;
    std::cout<<(diff_prof_edge/1000000)<<"ms"<<std::endl;


    //Copying Device result data to Host memory
    q.enqueueReadBuffer(imageFromDeviceedge, CL_TRUE, 0,
(height*width),out_img_edge.data);
    q.finish();
```

Top level kernel

Below is the top-level/wrapper function with all necessary glue logic.

```
// streaming based kernel
#include "xf_canny_config.h"

extern "C" {
void canny_accel(ap_uint<INPUT_PTR_WIDTH> *img_inp,
ap_uint<OUTPUT_PTR_WIDTH> *img_out, int rows, int cols,int
low_threshold,int high_threshold)
{
#pragma HLS INTERFACE m_axi     port=img_inp  offset=slave bundle=gmem1
#pragma HLS INTERFACE m_axi     port=img_out  offset=slave bundle=gmem2
#pragma HLS INTERFACE s_axilite port=img_inp  bundle=control
#pragma HLS INTERFACE s_axilite port=img_out  bundle=control

#pragma HLS INTERFACE s_axilite port=rows     bundle=control
#pragma HLS INTERFACE s_axilite port=cols     bundle=control
#pragma HLS INTERFACE s_axilite port=low_threshold     bundle=control
#pragma HLS INTERFACE s_axilite port=high_threshold     bundle=control
#pragma HLS INTERFACE s_axilite port=return   bundle=control

    xf::Mat<XF_8UC1, HEIGHT, WIDTH, INTYPE> in_mat(rows,cols);
#pragma HLS stream variable=in_mat.data depth=2

    xf::Mat<XF_2UC1, HEIGHT, WIDTH, XF_NPPC32> dst_mat(rows,cols);
#pragma HLS stream variable=dst_mat.data depth=2


    #pragma HLS DATAFLOW


xf::Array2xfMat<INPUT_PTR_WIDTH,XF_8UC1,HEIGHT,WIDTH,INTYPE>(img_inp,in_mat)
;
    xf::Canny<FILTER_WIDTH,NORM_TYPE,XF_8UC1,XF_2UC1,HEIGHT,
WIDTH,INTYPE,XF_NPPC32,XF_USE_URAM>(in_mat,dst_mat,low_threshold,high_thresh
old);

xf::xfMat2Array<OUTPUT_PTR_WIDTH,XF_2UC1,HEIGHT,WIDTH,XF_NPPC32>(dst_mat,img
_out);


}
}
// memory mapped kernel
#include "xf_canny_config.h"
extern "C" {
void edgetracing_accel(ap_uint<INPUT_PTR_WIDTH> *img_inp,
ap_uint<OUTPUT_PTR_WIDTH> *img_out, int rows, int cols)
{
#pragma HLS INTERFACE m_axi     port=img_inp  offset=slave bundle=gmem3
#pragma HLS INTERFACE m_axi     port=img_out  offset=slave bundle=gmem4
```

```
#pragma HLS INTERFACE s_axilite port=img_inp  bundle=control
#pragma HLS INTERFACE s_axilite port=img_out  bundle=control

#pragma HLS INTERFACE s_axilite port=rows     bundle=control
#pragma HLS INTERFACE s_axilite port=cols     bundle=control
#pragma HLS INTERFACE s_axilite port=return   bundle=control


    xf::Mat<XF_2UC1, HEIGHT, WIDTH, XF_NPPC32> _dst1(rows,cols,img_inp);
    xf::Mat<XF_8UC1, HEIGHT, WIDTH, XF_NPPC8> _dst2(rows,cols,img_out);
    xf::EdgeTracing<XF_2UC1,XF_8UC1,HEIGHT, WIDTH,
XF_NPPC32,XF_NPPC8,XF_USE_URAM>(_dst1,_dst2);

}
}
```

Evaluating the Functionality

You can build the kernels and test the functionality through software emulation, hardware emulation, and running directly on a supported hardware with the FPGA. For PCIe based platforms, use the following commands to setup the environment:

```
$ cd <path to the proj folder, where makefile is present>
$ source <path to the SDx installation folder>/SDx/<version number>/
settings64.sh
$ source <path to Xilinx_xrt>/packages/setenv.sh
$ export PLATFORM_PATH=<path to the platform folder>
$ export XLNX_SRC_PATH=<path to the xfOpenCV repo>
$ export XILINX_CL_PATH=/usr
```

Software Emulation

Software emulation is equivalent to running a C-simulation of the kernel. The time for compilation is minimal, and is therefore recommended to be the first step in testing the kernel. Following are the steps to build and run for the software emulation:

```
$ make all TARGETS=sw_emu
$ export XCL_EMULATION_MODE=sw_emu
$ export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:<sdx installation path>/SDx/
2019.1/lnx64/tools/opencv:/usr/lib64
$ ./<executable> <args>
```

Hardware Emulation

Hardware emulation runs the test on the generated RTL after synthesis of the C/C++ code. The simulation, since being done on RTL requires longer to complete when compared to software emulation. Following are the steps to build and run for the hardware emulation:

```
$ make all TARGETS=hw_emu
$ export XCL_EMULATION_MODE=hw_emu
$ export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:<sdx installation path>/SDx/
2019.1/lnx64/tools/opencv:/usr/lib64
$ ./<executable> <args>
```

Testing on the Hardware

To test on the hardware, the kernel must be compiled into a bitstream (building for hardware).

```
$ make all TARGETS=hw
```

This would consume some time since the C/C++ code must be converted to RTL, run through synthesis and implementation process before a bitstream is created. As a prerequisite the drivers has to be installed for corresponding DSA, for which the example was built for. Following are the steps to run the kernel on a hardware:

```
$ source /opt/xilinx/xrt/setup.sh
$ export XILINX_XRT=/opt/xilinx/xrt
$ cd <path to the executable and the corresponding xclbin>
$ ./<executable> <args>
```

Chapter 4

Getting Started with HLS

The xfOpenCV library can be used to build applications in Vivado® HLS. This section provides details on how the xfOpenCV library components can be integrated into a design in Vivado HLS 2019.1. This section of the document provides steps on how to run a single library component through the Vivado HLS 2019.1 use flow which includes, C-simulation, C-synthesis, C/RTL co-simulation, and exporting the RTL as an IP.

You are required to do the following changes to facilitate proper functioning of the use model in Vivado HLS 2019.1:

1. *Use of appropriate compile-time options* - When using the xfOpenCV functions in HLS, the `-D__SDSVHLS__` and `-std=c++0x` options need to be provided at the time of compilation:
2. *Specifying interface pragmas to the interface level arguments* - For the functions with top level interface arguments as pointers (with more than one read/write access), the `m_axi` Interface pragma must be specified. For example,

```
void lut_accel(xf::Mat<TYPE, HEIGHT, WIDTH, NPC1> &imgInput,
xf::Mat<TYPE, HEIGHT, WIDTH, NPC1> &imgOutput, unsigned char *lut_ptr)
{
#pragma HLS INTERFACE m_axi depth=256 port=lut_ptr offset=direct
bundle=lut_ptr
    xf::LUT< TYPE, HEIGHT, WIDTH, NPC1> (imgInput,imgOutput,lut_ptr);
}
```

HLS Standalone Mode

The HLS standalone mode can be operated using the following two modes:

1. TCL Script Mode
2. GUI Mode

Tcl Script Mode

Use the following steps to operate the HLS Standalone Mode using Tcl Script:

1. In the Vivado® HLS tcl script file, update the `cflags` in all the `add_files` sections.

2. Append the path to the `xfOpenCV/include` directory, as it contains all the header files required by the library.
3. Add the `-D__SDSVHLS__` and `-std=c++0x` compiler flags.

Note: When using Vivado HLS in the Windows operating system, provide the `-std=c++0x` flag only for C-Sim and Co-Sim. Do not include the flag when performing synthesis.

For example:

Setting flags for source files:

```
add_files xf_dilation_accel.cpp -cflags "-I<path-to-include-directory> -
D__SDSVHLS__ -std=c++0x"
```

Setting flags for testbench files:

```
add_files -tb xf_dilation_tb.cpp -cflags "-I<path-to-include-directory> -
D__SDSVHLS__ -std=c++0x"
```

GUI Mode

Use the following steps to operate the HLS Standalone Mode using GUI:

1. Open Vivado® HLS in GUI mode and create a new project
2. Specify the name of the project. For example - *Dilation*.
3. Click **Browse** to enter a workspace folder used to store your projects.
4. Click **Next**.
5. Under the source files section, add the `accel.cpp` file which can be found in the `examples` folder. Also, fill the top function name (here it is dilation_accel).
6. Click **Next**.
7. Under the test bench section add `tb.cpp`.
8. Click **Next**.
9. Select the clock period to the required value (10ns in example).
10. Select the suitable part. For example, `xczu9eg-ffvb1156-2-i`.
11. Click **Finish**.
12. Right click on the created project and select **Project Settings**.
13. In the opened tab, select **Simulation**.
14. Files added under the Test Bench section will be displayed. Select a file and click **Edit CFLAGS**.
15. Enter `-I<path-to-include-directory> -D__SDSVHLS__ -std=c++0x`.

> ***Note***: When using Vivado HLS in the Windows operating system, make sure to provide the `-std=c++0x` flag only for C-Sim and Co-Sim. Do not include the flag when performing synthesis.

16. Select **Synthesis** and repeat the above step for all the displayed files.
17. Click **OK**.
18. Run the C Simulation, select **Clean Build** and specify the required input arguments.
19. Click **OK**.
20. All the generated output files/images will be present in the `solution1->csim->build`.
21. Run `C Synthesis`.
22. Run `Co-simulation` by specifying the proper input arguments.
23. The status of co-simulation can be observed on the console.

Constraints for Co-simulation

There are few limitations in performing co-simulation of the xfOpenCV functions. They are:

1. Functions with multiple accelerators are not supported.
2. Compiler and simulator are default in HLS. (gcc, xsim).
3. Since HLS does not support multi-kernel integration, the current flow also does not support multi-kernel integration. Hence, the Pyramidal Optical flow and Canny Edge Detection functions and examples are not supported in this flow:
4. The maximum image size (HEIGHT and WIDTH) set in config.h file should be equal to the actual input image size.

AXI Video Interface Functions

xfOpenCV has functions that will transform the xf::Mat into Xilinx® Video Streaming interface and vice-versa. `xf::AXIvideo2xfMat()` and `xf::xfMat2AXIVideo()` act as video interfaces to the IPs of the xfOpenCV functions in the Vivado® IP integrator. `cvMat2AXIvideoxf <NPC>` and `AXIvideo2cvMatxf<NPC>` are used on the host side.

Table 15: **AXI Video Interface Functions**

Video Library Function	Description
AXIvideo2xfMat	Converts data from an AXI4 video stream representation to xf::Mat format.
xfMat2AXIvideo	Converts data stored as xf::Mat format to an AXI4 video stream.
cvMat2AXIvideoxf	Converts data stored as cv::Mat format to an AXI4 video stream

Table 15: **AXI Video Interface Functions** *(cont'd)*

Video Library Function	Description
AXIvideo2cvMatxf	Converts data from an AXI4 video stream representation to cv::Mat format.

AXIvideo2xfMat

The `AXIvideo2xfMat` function receives a sequence of images using the AXI4 Streaming Video and produces an `xf::Mat` representation.

API Syntax

```
template<int W,int T,int ROWS, int COLS,int NPC>
int AXIvideo2xfMat(hls::stream< ap_axiu<W,1,1,1> >& AXI_video_strm,
xf::Mat<T,ROWS, COLS, NPC>& img)
```

Parameter Descriptions

The following table describes the template and the function parameters.

Table 16: **AXIvideo2cvMatxf Function Parameter Description**

Parameter	Description
W	Data width of AXI4-Stream. Recommended value is pixel depth.
T	Pixel type of the image. 1 channel (XF_8UC1). Data width of pixel must be no greater than *W*.
ROWS	Maximum height of input image.
COLS	Maximum width of input image.
NPC	Number of pixels to be processed per cycle. Possible options are XF_NPPC1 and XF_NPPC8 for 1-pixel and 8-pixel operations respectively.
AXI_video_strm	HLS stream of ap_axiu (axi protocol) type.
img	Input image.

This function will return bit error of ERROR_IO_EOL_EARLY(1) or ERROR_IO_EOL_LATE(2) to indicate an unexpected line length, by detecting TLAST input.

For more information about AXI interface see UG761.

xfMat2AXIvideo

The `Mat2AXI` video function receives an xf::Mat representation of a sequence of images and encodes it correctly using the AXI4 Streaming video protocol.

API Syntax

```
template<int W, int T, int ROWS, int COLS,int NPC>
int xfMat2AXIvideo(xf::Mat<T,ROWS, COLS,NPC>&
img,hls::stream<ap_axiu<W,1,1,1> >& AXI_video_strm)
```

Parameter Descriptions

The following table describes the template and the function parameters.

Table 17: **xfMat2AXIvideo Function Parameter Description**

Parameter	Description
W	Data width of AXI4-Stream. Recommended value is pixel depth.
T	Pixel type of the image. 1 channel (XF_8UC1). Data width of pixel must be no greater than *W*.
ROWS	Maximum height of input image.
COLS	Maximum width of input image.
NPC	Number of pixels to be processed per cycle. Possible options are XF_NPPC1 and XF_NPPC8 for 1-pixel and 8-pixel operations respectively.
AXI_video_strm	HLS stream of ap_axiu (axi protocol) type.
img	Output image.

This function returns the value 0.

Note: The NPC values across all the functions in a data flow must follow the same value. If there is mismatch it throws a compilation error in HLS.

cvMat2AXIvideoxf

The `cvMat2Axivideoxf` function receives image as cv::Mat representation and produces the AXI4 streaming video of image.

API Syntax

```
template<int NPC,int W>
void cvMat2AXIvideoxf(cv::Mat& cv_mat, hls::stream<ap_axiu<W,1,1,1> >&
AXI_video_strm)
```

Parameter Descriptions

The following table describes the template and the function parameters.

Table 18: **AXIvideo2cvMatxf Function Parameter Description**

Parameter	Description
W	Data width of AXI4-Stream. Recommended value is pixel depth.
NPC	Number of pixels to be processed per cycle. Possible options are XF_NPPC1 and XF_NPPC8 for 1-pixel and 8-pixel operations respectively.
AXI_video_strm	HLS stream of ap_axiu (axi protocol) type.
cv_mat	Input image.

AXIvideo2cvMatxf

The `Axivideo2cvMatxf` function receives image as AXI4 streaming video and produces the cv::Mat representation of image

API Syntax

```
template<int NPC,int W>
void AXIvideo2cvMatxf(hls::stream<ap_axiu<W,1,1,1> >& AXI_video_strm,
cv::Mat& cv_mat)
```

Parameter Descriptions

The following table describes the template and the function parameters.

Table 19: **AXIvideo2cvMatxf Function Parameter Description**

Parameter	Description
W	Data width of AXI4-Stream. Recommended value is pixel depth.
NPC	Number of pixels to be processed per cycle. Possible options are XF_NPPC1 and XF_NPPC8 for 1-pixel and 8-pixel operations respectively.
AXI_video_strm	HLS stream of ap_axiu (axi protocol) type.
cv_mat	Output image.

xfOpenCV Library API Reference

To facilitate local memory allocation on FPGA devices, the xfOpenCV library functions are provided in templates with compile-time parameters. Data is explicitly copied from `cv::Mat` to `xf::Mat` and is stored in physically contiguous memory to achieve the best possible performance. After processing, the output in `xf::Mat` is copied back to `cv::Mat` to write it into the memory.

xf::Mat Image Container Class

`xf::Mat` is a template class that serves as a container for storing image data and its attributes.

Note: The `xf::Mat` image container class is similar to the `cv::Mat` class of the OpenCV library.

Class Definition

```
template<int T, int ROWS, int COLS, int NPC>
class Mat {

  public:
    unsigned char allocatedFlag;            // flag to mark memory
allocation in this class
    int rows, cols, size;                   // actual image size

#ifdef __SDSVHLS__
    typedef XF_TNAME(T,NPC) DATATYPE;
#else                                       // When not being built for V-
HLS
    typedef struct {
        XF_CTUNAME(T,NPC) chnl[XF_NPIXPERCYCLE(NPC)][XF_CHANNELS(T,NPC)];
    } __attribute__ ((packed)) DATATYPE;
#endif

//#if (defined  (__SDSCC__) ) || (defined (__SYNTHESIS__))
#if defined (__SYNTHESIS__) && !defined (__SDA_MEM_MAP__)
    DATATYPE *data __attribute((xcl_array_geometry((ROWS)*(COLS>>
(XF_BITSHIFT(NPC))))));//data[ ROWS * ( COLS >> ( XF_BITSHIFT ( NPC ) ) ) ];
#else
    DATATYPE *data;
#endif


    Mat();                                  // default constructor
    Mat(Size _sz);
    Mat(int _rows, int _cols);
```

```
    Mat(int _size, int _rows, int _cols);
    Mat(int _rows, int _cols, void *_data);
    Mat(const Mat&);                        // copy constructor

    ~Mat();

    Mat& operator= (const Mat&);            // Assignment operator
//  XF_TNAME(T, XF_NPPC1) operator() (unsigned int r, unsigned int c);
//  XF_CTUNAME(T, NPC) operator() (unsigned int r, unsigned int c, unsigned
int ch);
    XF_TNAME(T,NPC) read(int index);
    float read_float(int index);
    void write(int index, XF_TNAME(T,NPC) val);
    void write_float(int index, float val);

    void init (int _rows, int _cols, bool allocate=true);
    void copyTo (void* fromData);
    unsigned char* copyFrom ();

    const int type() const;
    const int depth() const;
    const int channels() const;

    template<int DST_T>
    void convertTo (Mat<DST_T, ROWS, COLS, NPC> &dst, int otype, double
alpha=1, double beta=0);
};
```

Parameter Descriptions

The following table lists the `xf::Mat`

Table 20: **xf::Mat Class Parameter Descriptions**

Parameter	Description
rows	The number of rows in the image or height of the image.
cols	The number of columns in the image or width of the image. class parameters and their descriptions:
size	The number of words stored in the data member. The value is calculated using `rows*cols/(number of pixels packed per word)`.
allocatedFlag	Flag for memory allocation status
*data	class parameters and the pointer to the words that store the pixels of the image.

Member Functions Description

The following table lists the member functions and their descriptions:

Table 21: **xf::Mat Member Function Descriptions**

Member Functions	Description
Mat()	This default constructor initializes the Mat object sizes, using the template parameters ROWS and COLS.
Mat(int _rows, int _cols)	This constructor initializes the Mat object using arguments _rows and _cols.
Mat(const xf::Mat &_src)	This constructor helps clone a Mat object to another. New memory will be allocated for the newly created constructor.
Mat(int _rows, int _cols, void *_data)	This constructor initializes the Mat object using arguments _rows, _cols, and _data. The *data member of the Mat object points to the memory allocated for _data argument, when this constructor is used. No new memory is allocated for the *data member.
convertTo(Mat<DST_T,ROWS, COLS, NPC> &dst, int otype, double alpha=1, double beta=0)	Refer to xf::convertTo
copyTo(* fromData)	Copies the data from Data pointer into physically contiguous memory allocated inside the constructor.
copyFrom()	Returns the pointer to the first location of the *data member.
read(int index)	Readout a value from a given location and return it as a packed (for multi-pixel/clock) value.
read_float(int index)	Readout a value from a given location and return it as a float value
write(int index, XF_TNAME(T,NPC) val)	Writes a packed (for multi-pixel/clock) value into the given location.
write_float(int index, float val)	Writes a float value into the given location.
type()	Returns the type of the image.
depth()	Returns the depth of the image
channels()	Returns number of channels of the image
~Mat()	This is a default destructor of the Mat object.

Template Parameter Descriptions

Template parameters of the `xf::Mat` class are used to set the depth of the pixel, number of channels in the image, number of pixels packed per word, maximum number of rows and columns of the image. The following table lists the template parameters and their descriptions:

Table 22: **xf::Mat Template Parameter Descriptions**

Parameters	Description
TYPE	Type of the pixel data. For example, XF_8UC1 stands for 8-bit unsigned and one channel pixel. More types can be found in `include/common/xf_params.h`.
HEIGHT	Maximum height of an image.
WIDTH	Maximum width of an image.
NPC	The number of pixels to be packed per word. For instance, XF_NPPC1 for 1 pixel per word; and XF_NPPC8 for 8 pixels per word.

Pixel-Level Parallelism

The amount of parallelism to be implemented in a function from xfOpenCV is kept as a configurable parameter. In most functions, there are two options for processing data.

- Single-pixel processing
- Processing eight pixels in parallel

The following table describes the options available for specifying the level of parallelism required in a particular function:

Table 23: **Options Available for Specifying the Level of Parallelism**

Option	Description
XF_NPPC1	Process 1 pixel per clock cycle
XF_NPPC2	Process 2 pixels per clock cycle
XF_NPPC4	Process 4 pixels per clock cycle
XF_NPPC8	Process 8 pixels per clock cycle

Macros to Work With Parallelism

There are two macros that are defined to work with parallelism.

- The `XF_NPIXPERCYCLE(flags)` macro resolves to the number of pixels processed per cycle.
  - `XF_NPIXPERCYCLE(XF_NPPC1)` resolves to 1
  - `XF_NPIXPERCYCLE(XF_NPPC2)` resolves to 2
  - `XF_NPIXPERCYCLE(XF_NPPC4)` resolves to 4

  - `XF_NPIXPERCYCLE(XF_NPPC8)` resolves to 8
- The `XF_BITSHIFT(flags)` macro resolves to the number of times to shift the image size to right to arrive at the final data transfer size for parallel processing.
  - `XF_BITSHIFT(XF_NPPC1)` resolves to 0
  - `XF_BITSHIFT(XF_NPPC2)` resolves to 1
  - `XF_BITSHIFT(XF_NPPC4)` resolves to 2
  - `XF_BITSHIFT(XF_NPPC8)` resolves to 3

Pixel Types

Parameter types will differ, depending on the combination of the depth of pixels and the number of channels in the image. The generic nomenclature of the parameter is listed below.

```
XF_<Number of bits per pixel><signed (S) or unsigned (U) or float
(F)>C<number of channels>
```

For example, for an 8-bit pixel - unsigned - 1 channel the data type is `XF_8UC1`.

The following table lists the available data types for the `xf::Mat` class:

Table 24: **xf::Mat Class - Available Data Types**

Option	Number of bits per Pixel	Unsigned/ Signed/ Float Type	Number of Channels
XF_8UC1	8	Unsigned	1
XF_16UC1	16	Unsigned	1
XF_16SC1	16	Signed	1
XF_32UC1	32	Unsigned	1
XF_32FC1	32	Float	1
XF_32SC1	32	Signed	1
XF_8UC2	8	Unsigned	2
XF_8UC4	8	Unsigned	4
XF_8UC3	8	Unsigned	3
XF_2UC1	2	Unsigned	1

Manipulating Data Type

Based on the number of pixels to process per clock cycle and the type parameter, there are different possible data types. The xfOpenCV library uses these datatypes for internal processing and inside the `xf::Mat` class. The following are a few supported types:

- `XF_TNAME(TYPE,NPPC)` resolves to the data type of the data member of the `xf::Mat` object. For instance, `XF_TNAME(XF_8UC1,XF_NPPC8)` resolves to `ap_uint<64>`.

- `Word width = pixel depth * number of channels * number of pixels to process per cycle (NPPC).`
- `XF_DTUNAME(TYPE,NPPC)` resolves to the data type of the pixel. For instance, `XF_DTUNAME(XF_32FC1,XF_NPPC1)` resolves to `float`.
- `XF_PTSNAME(TYPE,NPPC)` resolves to the 'C' data type of the pixel. For instance, `XF_PTSNAME (XF_16UC1,XF_NPPC2)` resolves to `unsigned short`.

Note: `ap_uint<>`, `ap_int<>`, `ap_fixed<>`, and `ap_ufixed<>` types belong to the high-level synthesis (HLS) library. For more information, see the *Vivado Design Suite User Guide: High-Level Synthesis* (UG902).

Sample Illustration

The following code illustrates the configurations that are required to build the gaussian filter on an image, using the SDSoC™ tool for Zynq® UltraScale™ platform.

Note: In case of a real-time application, where the video is streamed in, it is recommended that the location of frame buffer is `xf::Mat` and is processed using the library function. The resultant location pointer is passed to display IPs.

`xf_config_params.h`

```
#define FILTER_SIZE_3 1
#define FILTER_SIZE_5 0
#define FILTER_SIZE_7 0
#define RO 0
#define NO 1

#if NO
#define NPC1 XF_NPPC1
#endif
#if RO
#define NPC1 XF_NPPC8
#endif
```

`xf_gaussian_filter_tb.cpp`

```
int main(int argc, char **argv)
{
cv::Mat in_img, out_img, ocv_ref;
cv::Mat in_gray, in_gray1, diff;
in_img = cv::imread(argv[1], 1); // reading in the color image
        extractChannel(in_img, in_gray, 1);

xf::Mat<XF_8UC1, HEIGHT, WIDTH, NPC1> imgInput(in_img.rows,in_img.cols);
xf::Mat<XF_8UC1, HEIGHT, WIDTH, NPC1> imgOutput(in_img.rows,in_img.cols);

imgInput.copyTo(in_gray.data);

gaussian_filter_accel(imgInput,imgOutput,sigma);

// Write output image
xf::imwrite("hls_out.jpg",imgOutput);
}
```

`xf_gaussian_filter_accel.cpp`

```
#include "xf_gaussian_filter_config.h"

void gaussian_filter_accel(xf::Mat<XF_8UC1,HEIGHT,WIDTH,NPC1>
&imgInput,xf::Mat<XF_8UC1,HEIGHT,WIDTH,NPC1>&imgOutput,float sigma)
{
    xf::GaussianBlur<FILTER_WIDTH, XF_BORDER_CONSTANT, XF_8UC1, HEIGHT,
WIDTH, NPC1>(imgInput, imgOutput, sigma);
}
```

`xf_gaussian_filter.hpp`

```
#pragma SDS data data_mover("_src.data":AXIDMA_SIMPLE)
        #pragma SDS data data_mover("_dst.data":AXIDMA_SIMPLE)
        #pragma SDS data access_pattern("_src.data":SEQUENTIAL)
        #pragma SDS data copy("_src.data"[0:"_src.size"])
        #pragma SDS data access_pattern("_dst.data":SEQUENTIAL)
        #pragma SDS data copy("_dst.data"[0:"_dst.size"])

        template<int FILTER_SIZE, int BORDER_TYPE, int SRC_T, int ROWS, int
COLS,int NPC = 1>
        void GaussianBlur(xf::Mat<SRC_T, ROWS, COLS, NPC> & _src,
xf::Mat<SRC_T, ROWS, COLS, NPC> & _dst, float sigma)
        {
        //function body
        }
```

The design fetches data from external memory (with the help of SDSoC data movers) and is transferred to the function in 8-bit or 64-bit packets, based on the configured mode. Assuming 8-bits per pixel, 8 pixels can be packed into 64-bits. Therefore, 8 pixels are available to be processed in parallel.

Enable the `FILTER_SIZE_3` and the `NO` macros in the `xf_config_params.h` file. The macro is used to set the filter size to `3x3` and `#define NO 1` macro enables 1 pixel parallelism.

Specify the SDSoC tool specific pragmas, in the `xf_gaussian_filter.hpp` file.

```
#pragma SDS data data_mover("_src.data":AXIDMA_SIMPLE)
#pragma SDS data data_mover("_dst.data":AXIDMA_SIMPLE)
#pragma SDS data access_pattern("_src.data":SEQUENTIAL)
#pragma SDS data copy("_src.data"[0:"_src.size"])
#pragma SDS data access_pattern("_dst.data":SEQUENTIAL)
#pragma SDS data copy("_dst.data"[0:"_dst.size"])
```

Note: For more information on the pragmas used for hardware accelerator functions in SDSoC, see *SDSoC Environment User Guide* (UG1027).

Additional Utility Functions for Software

xf::imread

The function xf::imread loads an image from the specified file path, copies into xf::Mat and returns it. If the image cannot be read (because of missing file, improper permissions, unsupported or invalid format), the function exits with a non-zero return code and an error statement.

Note: In an HLS standalone mode like Cosim, use `cv::imread` followed by `copyTo` function, instead of `xf::imread`.

API Syntax

```
template<int PTYPE, int ROWS, int COLS, int NPC>
xf::Mat<PTYPE, ROWS, COLS, NPC> imread (char *filename, int type)
```

Parameter Descriptions

The table below describes the template and the function parameters.

Table 25: **xf::imread Function Parameter Descriptions**

Parameter	Description
PTYPE	Input pixel type. Value should be in accordance with the 'type' argument's value.
ROWS	Maximum height of the image to be read
COLS	Maximum width of the image to be read
NPC	Number of pixels to be processed per cycle; possible options are XF_NPPC1 and XF_NPPC8 for 1 pixel and 8 pixel operations respectively.
filename	Name of the file to be loaded
type	Flag that depicts the type of image. The values are: • '0' for gray scale • '1' for color image

xf::imwrite

The function xf::imwrite saves the image to the specified file from the given xf::Mat. The image format is chosen based on the file name extension. This function internally uses cv::imwrite for the processing. Therefore, all the limitations of cv::imwrite are also applicable to xf::imwrite.

API Syntax

```
template <int PTYPE, int ROWS, int COLS, int NPC>
void imwrite(const char *img_name, xf::Mat<PTYPE, ROWS, COLS, NPC> &img)
```

Parameter Descriptions

The table below describes the template and the function parameters.

Table 26: **xf::imwrite Function Parameter Descriptions**

Parameter	Description
PTYPE	Input pixel type. Supported types are: XF_8UC1, XF_16UC1, XF_8UC4, and XF_16UC4
ROWS	Maximum height of the image to be read
COLS	Maximum width of the image to be read
NPC	Number of pixels to be processed per cycle; possible options are XF_NPPC1 and XF_NPPC8 for 1 pixel and 8 pixel operations respectively.
img_name	Name of the file with the extension
img	xf::Mat array to be saved

xf::absDiff

The function xf::absDiff computes the absolute difference between each individual pixels of an xf::Mat and a cv::Mat, and returns the difference values in a cv::Mat.

API Syntax

```
template <int PTYPE, int ROWS, int COLS, int NPC>
void absDiff(cv::Mat &cv_img, xf::Mat<PTYPE, ROWS, COLS, NPC>& xf_img,
cv::Mat &diff_img )
```

Parameter Descriptions

The table below describes the template and the function parameters.

Table 27: **xf::absDiff Function Parameter Descriptions**

Parameter	Description
PTYPE	Input pixel type
ROWS	Maximum height of the image to be read
COLS	Maximum width of the image to be read
NPC	Number of pixels to be processed per cycle; possible options are XF_NPPC1, XF_NPPC4, and XF_NPPC8 for 1-pixel, 4-pixel, and 8-pixel parallel operations respectively.
cv_img	cv::Mat array to be compared
xf_img	xf::Mat array to be compared
diff_img	Output difference image(cv::Mat)

xf::convertTo

The xf::convertTo function performs bit depth conversion on each individual pixel of the given input image. This method converts the source pixel values to the target data type with appropriate casting.

```
dst(x,y)= cast<target-data-type>(α(src(x,y)+β))
```

Note: The output and input Mat cannot be the same. That is, the converted image cannot be stored in the Mat of the input image.

API Syntax

```
template<int DST_T> void convertTo(xf::Mat<DST_T,ROWS, COLS, NPC> &dst, int
ctype, double alpha=1, double beta=0)
```

Parameter Descriptions

The table below describes the template and function parameters.

Table 28: **xf::convertTo Function Parameter Descriptions**

Parameter	Description
DST_T	Output pixel type. Possible values are XF_8UC1, XF_16UC1, XF_16SC1, and XF_32SC1.
ROWS	Maximum height of image to be read
COLS	Maximum width of image to be read
NPC	Number of pixels to be processed per cycle; possible options are XF_NPPC1, XF_NPPC4, and XF_NPPC8 for 1-pixel, 4-pixel, and 8-pixel parallel operations respectively. XF_32SC1 and XF_NPPC8 combination is not supported.
dst	Converted xf Mat

ctype	Conversion type : Possible values are listed here.
	//Down-convert:
	• XF_CONVERT_16U_TO_8U
	• XF_CONVERT_16S_TO_8U
	• XF_CONVERT_32S_TO_8U
	• XF_CONVERT_32S_TO_16U
	• XF_CONVERT_32S_TO_16S
	//Up-convert:
	• XF_CONVERT_8U_TO_16U
	• XF_CONVERT_8U_TO_16S
	• XF_CONVERT_8U_TO_32S
	• XF_CONVERT_16U_TO_32S
	• XF_CONVERT_16S_TO_32S

Table 28: **xf::convertTo Function Parameter Descriptions** *(cont'd)*

Parameter	Description
alpha	Optional scale factor
beta	Optional delta added to the scaled values

xfOpenCV Library Functions

The xfOpenCV library is a set of select OpenCV functions optimized for Zynq-7000 and Zynq UltraScale+ MPSoC devices. The following table lists the xfOpenCV library functions.

Table 29: **xfOpenCV Library Functions**

Computations	Input Processing	Filters	Other
Absolute Difference	Bit Depth Conversion	Bilateral Filter	Canny Edge Detection
Accumulate	Channel Combine	Box Filter	FAST Corner Detection
Accumulate Squared	Channel Extract	Custom Convolution	Harris Corner Detection
Accumulate Weighted	Color Conversion	Dilate	Histogram Computation
Atan2	Histogram Equalization	Erode	Dense Pyramidal LK Optical Flow
Bitwise AND, Bitwise NOT, Bitwise OR, Bitwise XOR	Look Up Table	Gaussian Filter	Dense Non-Pyramidal LK Optical Flow
Gradient Magnitude	Remap	Sobel Filter	MinMax Location
Gradient Phase	Resolution Conversion (Resize)	Median Blur Filter	Thresholding
Integral Image	convertScaleAbs	Scharr Filter	SVM
Inverse (Reciprocal)	Demosaicing		Otsu Threshold
Pixel-Wise Addition	Crop		Mean Shift Tracking
Pixel-Wise Multiplication	Reduce		HOG
Pixel-Wise Subtraction	BoundingBox		Stereo Local Block Matching
Square Root			WarpTransform
Mean and Standard Deviation			Pyramid Up
AddS, Compare, CompareS, Max, MaxS, Min, MinS, Set, SubRS, SubS, Zero			Pyramid Down
Sum			Delay
Addweighted			Duplicate
			Color Thresholding
			BGR2HSV
			InitUndistortRectifyMapInverse

Table 29: **xfOpenCV Library Functions** *(cont'd)*

Computations	Input Processing	Filters	Other
			HoughLines
			Semi Global Method for Stereo Disparity Estimation
			Paintmask
			InRange
			Kalman Filter

Notes:

1. The maximum resolution supported for all the functions is 4K, except Houghlines and HOG (RB mode).

Note: Resolution Conversion (Resize) in 8 pixel per cycle mode, Dense Pyramidal LK Optical Flow, and Dense Non-Pyramidal LK Optical Flow functions are not supported on the Zynq-7000 SoC ZC702 devices, due to the higher resource utilization.

Note: Number of pixel per clock depends on the maximum bus width a device can support.

For example: Zynq-7000 Soc has 64 bit interface and so for a pixel type 16UC1 ,maximum of four pixel per clock(XF_NPPC4) is possible.

Absolute Difference

The `absdiff` function finds the pixel wise absolute difference between two input images and returns an output image. The input and the output images must be the XF_8UC1 type.

$$I_{out}(x, y) = |I_{in1}(x, y) - I_{in2}(x, y)|$$

Where,

- $I_{out}(x, y)$ is the intensity of output image at (x,y) position.
- $I_{in1}(x, y)$ is the intensity of first input image at (x,y) position.
- $I_{in2}(x, y)$ is the intensity of second input image at (x,y) position.

API Syntax

```
template<int SRC_T, int ROWS, int COLS, int NPC=1>
void absdiff(
xf::Mat<int SRC_T, int ROWS, int COLS, int NPC> src1,
xf::Mat<int SRC_T, int ROWS, int COLS, int NPC> src2,
xf::Mat<int SRC_T, int ROWS, int COLS, int NPC> dst )
```

Parameter Descriptions

The following table describes the template and the function parameters.

Table 30: **absdiff Function Parameter Descriptions**

Parameter	Description
SRC_T	Input and Output pixel type. Only 8-bit, unsigned, 1 and 3 channels are supported (XF_8UC1 and XF_8UC3)
ROWS	Maximum height of input and output image.
COLS	Maximum width of input and output image. Must be multiple of 8, for 8-pixel operation.
NPC	Number of pixels to be processed per cycle; possible options are XF_NPPC1 and XF_NPPC8 for 1 pixel and 8 pixel operations respectively.
src1	Input image
src2	Input image
dst	Output image

Resource Utilization

The following table summarizes the resource utilization in different configurations, generated using Vivado HLS 2019.1 tool for the Xczu9eg-ffvb1156-1-i-es1 FPGA, to process a grayscale HD (1080x1920) image.

Table 31: **absdiff Function Resource Utilization Summary**

Operating Mode	Operating Frequency (MHz)	Utilization Estimate				
		BRAM_18K	DSP_48Es	FF	LUT	CLB
1 pixel	300	0	0	62	67	17
8 pixel	150	0	0	67	234	39

Performance Estimate

The following table summarizes the performance in different configurations, as generated using Vivado HLS 2019.1 tool for the Xczu9eg-ffvb1156-1-i-es1, to process a grayscale HD (1080x1920) image.

Table 32: **absdiff Function Performance Estimate Summary**

Operating Mode	Latency Estimate
	Max Latency (ms)
1 pixel operation (300 MHz)	6.9
8 pixel operation (150 MHz)	1.69

Deviation from OpenCV

There is no deviation from OpenCV, except that the `absdiff` function supports 8-bit pixels.

Accumulate

The `accumulate` function adds an image (src1) to the accumulator image (src2), and generates the accumulated result image (dst).

$$dst(x, y) = src1(x, y) + src2(x, y)$$

API Syntax

```
template<int SRC_T, int DST_T, int ROWS, int COLS, int NPC=1>
void accumulate (
xf::Mat<int SRC_T, int ROWS, int COLS, int NPC> src1,
xf::Mat<int SRC_T, int ROWS, int COLS, int NPC> src2,
xf::Mat<int DST_T, int ROWS, int COLS, int NPC> dst )
```

Parameter Descriptions

The following table describes the template and the function parameters.

Table 33: **accumulate Function Parameter Descriptions**

Parameter	Description
SRC_T	Input pixel type. Only 8-bit, unsigned, 1 and 3 channels are supported (XF_8UC1 and XF_8UC3)
DST_T	Output pixel type. Only 16-bit, unsigned, 1 and 3 channels are supported (XF_16UC1 and XF_16UC3)
ROWS	Maximum height of input and output image.
COLS	Maximum width of input and output image. Recommend using a multiple of 8, for an 8-pixel operation.
NPC	Number of pixels to be processed per cycle; possible options are XF_NPPC1 and XF_NPPC8 for 1 pixel and 8 pixel operations respectively.
src1	Input image
src2	Input image
dst	Output image

Resource Utilization

The following table summarizes the resource utilization in different configurations, generated using Vivado HLS 2019.1 tool for the Xczu9eg-ffvb1156-1-i-es1, to process a grayscale HD (1080x1920) image.

Table 34: **accumulate Function Resource Utilization Summary**

Operating Mode	Operating Frequency (MHz)	Utilization Estimate				
		BRAM_18K	DSP_48E	FF	LUT	CLB
1 pixel	300	0	0	62	55	12

Table 34: **accumulate Function Resource Utilization Summary** *(cont'd)*

Operating Mode	Operating Frequency (MHz)	Utilization Estimate				
		BRAM_18K	DSP_48E	FF	LUT	CLB
8 pixel	150	0	0	389	285	61

The following table summarizes the resource utilization in different configurations, generated using Vivado HLS 2019.1 tool for the Xczu9eg-ffvb1156-1-i-es1, to process 4K 3 Channel image.

Table 35: **accumulate Function Resource Utilization Summary**

Operating Mode	Operating Frequency (MHz)	Utilization Estimate				
		BRAM_18K	DSP_48E	FF	LUT	CLB
1 pixel	300	0	1	207	72	32

Performance Estimate

The following table summarizes the performance in different configurations, as generated using Vivado HLS 2019.1 tool for the Xczu9eg-ffvb1156-1-i-es1, to process a grayscale HD (1080x1920) image.

Table 36: **accumulate Function Performance Estimate Summary**

Operating Mode	Latency Estimate
	Max Latency (ms)
1 pixel operation (300 MHz)	6.9
8 pixel operation (150 MHz)	1.7

Deviation from OpenCV

In OpenCV the accumulated image is stored in the second input image. The src2 image acts as both input and output, as shown below:

$$src2(x,\ y) = src1(x,\ y) + src2(x,\ y)$$

Whereas, in the xfOpenCV implementation, the accumulated image is stored separately, as shown below:

$$dst(x,\ y) = src1(x,\ y) + src2(x,\ y)$$

Accumulate Squared

The `accumulateSquare` function adds the square of an image (src1) to the accumulator image (src2) and generates the accumulated result (dst).

$$dst(x, y) = src1(x, y)^2 + src2(x, y)$$

The accumulated result is a separate argument in the function, instead of having src2 as the accumulated result. In this implementation, having a bi-directional accumulator is not possible as the function makes use of streams.

API Syntax

```
template<int SRC_T, int DST_T, int ROWS, int COLS, int NPC=1>
void accumulateSquare (
xf::Mat<int SRC_T, int ROWS, int COLS, int NPC> src1,
xf::Mat<int SRC_T, int ROWS, int COLS, int NPC> src2,
xf::Mat<int DST_T, int ROWS, int COLS, int NPC> dst)
```

Parameter Descriptions

The following table describes the template and the function parameters.

Table 37: **accumulateSquare Function Parameter Descriptions**

Parameter	Description
SRC_T	Input pixel type. Only 8-bit, unsigned, 1 and 3 channels are supported (XF_8UC1 and XF_8UC3)
DST_T	Output pixel type. Only 16-bit, unsigned, 1 and 3 channels are supported (XF_16UC1 and XF_16UC3)
ROWS	Maximum height of input and output image.
COLS	Maximum width of input and output image (must be multiple of 8, for 8-pixel operation)
NPC	Number of pixels to be processed per cycle; possible options are XF_NPPC1 and XF_NPPC8 for 1 pixel and 8 pixel operations respectively.
src1	Input image
src2	Input image
dst	Output image

Resource Utilization

The following table summarizes the resource utilization in different configurations, generated using Vivado HLS 2019.1 tool for the Xczu9eg-ffvb1156-1-i-es1 FPGA, to process a grayscale HD (1080x1920) image.

Table 38: **accumulateSquare Function Resource Utilization Summary**

Operating Mode	Operating Frequency (MHz)	Utilization Estimate				
		BRAM_18K	DSP_48E	FF	LUT	CLB
1 pixel	300	0	1	71	52	14
8 pixel	150	0	8	401	247	48

The following table summarizes the resource utilization in different configurations, generated using Vivado HLS 2019.1 tool for the Xczu9eg-ffvb1156-1-i-es1 FPGA, to process 4K 3 Channel image.

Table 39: **accumulateSquare Function Resource Utilization Summary**

Operating Mode	Operating Frequency (MHz)	Utilization Estimate				
		BRAM_18K	DSP_48E	FF	LUT	CLB
1 pixel	300	0	3	227	86	37

Performance Estimate

The following table summarizes the performance in different configurations, as generated using Vivado HLS 2019.1 tool for the Xczu9eg-ffvb1156-1-i-es1, to process a grayscale HD (1080x1920) image.

Table 40: **accumulateSquare Function Performance Estimate Summary**

Operating Mode	Latency Estimate
	Max Latency (ms)
1 pixel operation (300 MHz)	6.9
8 pixel operation (150 MHz)	1.6

Deviation from OpenCV

In OpenCV the accumulated squared image is stored in the second input image. The src2 image acts as input as well as output.

$$src2(x,\ y) = src1(x,\ y)^2 + src2(x,\ y)$$

Whereas, in the xfOpenCV implementation, the accumulated squared image is stored separately.

$$dst(x,\ y) = src1(x,\ y)^2 + src2(x,\ y)$$

Accumulate Weighted

The `accumulateWeighted` function computes the weighted sum of the input image (src1) and the accumulator image (src2) and generates the result in dst.

$$dst(x, y) = alpha*src1(x, y) + (1 - alpha)*src2(x, y)$$

The accumulated result is a separate argument in the function, instead of having src2 as the accumulated result. In this implementation, having a bi-directional accumulator is not possible, as the function uses streams.

API Syntax

```
template<int SRC_T, int DST_T, int ROWS, int COLS, int NPC=1>
void accumulateWeighted (
xf::Mat<int SRC_T, int ROWS, int COLS, int NPC> src1,
xf::Mat<int SRC_T, int ROWS, int COLS, int NPC> src2,
xf::Mat<int DST_T, int ROWS, int COLS, int NPC> dst,
float alpha )
```

Parameter Descriptions

The following table describes the template and the function parameters.

Table 41: **accumulateWeighted Function Parameter Descriptions**

Parameter	Description
SRC_T	Input pixel type. Only 8-bit, unsigned, 1 and 3 channels are supported (XF_8UC1 and XF_8UC3)
DST_T	Output pixel type. Only 16-bit, unsigned, 1 and 3 channels are supported (XF_16UC1 and XF_16UC3)
ROWS	Maximum height of input and output image.
COLS	Maximum width of input and output image. Recommend multiples of 8, for an 8-pixel operation.
NPC	Number of pixels to be processed per cycle; possible options are XF_NPPC1 and XF_NPPC8 for 1 pixel and 8 pixel operations respectively.
src1	Input image
src2	Input image
dst	Output image
alpha	Weight applied to input image

Resource Utilization

The following table summarizes the resource utilization in different configurations, generated using Vivado HLS 2019.1 tool for the Xczu9eg-ffvb1156-1-i-es1 FPGA, to process a grayscale HD (1080x1920) image.

Table 42: **accumulateWeighted Function Resource Utilization Summary**

Operating Mode	Operating Frequency (MHz)	Utilization Estimate				
		BRAM_18K	DSP_48Es	FF	LUT	CLB
1 pixel	300	0	5	295	255	52
8 pixel	150	0	19	556	476	88

The following table summarizes the resource utilization in different configurations, generated using Vivado HLS 2019.1 tool for the Xczu9eg-ffvb1156-1-i-es1 FPGA, to process a 4K 3 Channel image.

Table 43: **accumulateWeighted Function Resource Utilization Summary**

Operating Mode	Operating Frequency (MHz)	Utilization Estimate				
		BRAM_18K	DSP_48Es	FF	LUT	CLB
1 pixel	300	0	9	457	387	95

Performance Estimate

The following table summarizes the performance in different configurations, as generated using Vivado HLS 2019.1 tool for the Xczu9eg-ffvb1156-1-i-es1, to process a grayscale HD (1080x1920) image.

Table 44: **accumulateWeighted Function Performance Estimate Summary**

Operating Mode	Latency Estimate
	Max Latency (ms)
1 pixel operation (300 MHz)	6.9
8 pixel operation (150 MHz)	1.7

Deviation from OpenCV

The resultant image in OpenCV is stored in the second input image. The src2 image acts as input as well as output, as shown below:

$$src2(x, y) = alpha*src1(x, y) + (1 - alpha)*src2(x, y)$$

Whereas, in xfOpenCV implementation, the accumulated weighted image is stored separately.

$$dst(x, y) = alpha*src1(x, y) + (1 - alpha)*src2(x, y)$$

AddS

The AddS function performs the addition operation between pixels of input image src and given scalar value scl and stores the result in dst.

$$dst(x,y) = src(x,y) + scl$$

Where (x,y) is the spatial coordinate of the pixel.

API Syntax

```
template<int POLICY_TYPE, int SRC_T, int ROWS, int COLS, int NPC =1>
void addS(xf::Mat<SRC_T, ROWS, COLS, NPC> & _src1, unsigned char
_scl[XF_CHANNELS(SRC_T,NPC)],xf::Mat<SRC_T, ROWS, COLS, NPC> & _dst)
```

Parameter Descriptions

The following table describes the template and the function parameters.

Table 45: **AddS Function Parameter Descriptions**

Parameter	Description
SRC_T	Input pixel type. 8-bit, unsigned, 1 channel is supported (XF_8UC1).
ROWS	Maximum height of input and output image.
COLS	Maximum width of input and output image. In case of N-pixel parallelism, width should be multiple of N.
NPC	Number of pixels to be processed per cycle; possible options are XF_NPPC1 and XF_NPPC8 for 1 pixel and 8 pixel operations respectively.
_src1	First input image
_scl	Input scalar value, the size should be number of channels.
_dst	Output image

Resource Utilization

The following table summarizes the resource utilization of the AddS function in both the resource optimized (8 pixel) mode and normal mode, as generated using Vivado HLS 2019.1 version tool for the Xczu9eg-ffvb1156-1-i-es1 FPGA.

Table 46: **AddS Function Resource Utilization Summary**

Name	Resource Utilization	
	1 pixel per clock operation	8 pixel per clock operation
	300 MHz	150 MHz
BRAM_18K	0	0
DSP48E	0	0
FF	100	101
LUT	52	185

Table 46: **AddS Function Resource Utilization Summary** *(cont'd)*

Name	Resource Utilization	
	1 pixel per clock operation	8 pixel per clock operation
	300 MHz	150 MHz
CLB	20	45

Performance Estimate

The following table summarizes a performance estimate of the kernel in different configurations, generated using Vivado HLS 2019.1 tool for Xczu9eg-ffvb1156-1-i-es1 FPGA to process a grayscale HD (1080x1920) image.

Table 47: **AddS Function Performance Estimate Summary**

Operating Mode	Latency Estimate	
	Operating Frequency (MHz)	Latency (ms)
1 pixel	300	6.9
8 pixel	150	1.7

Addweighted

The addweighted function calculates a weighted sum of two input images src1, src2 and generates the result in dst.

$$dst(x,y)= src1(x,y)*alpha+src2(x,y)*beta+ gamma$$

API Syntax

```
template< int SRC_T , int DST_T,   int ROWS, int COLS, int NPC=1>
void addWeighted(xf::Mat<SRC_T, ROWS, COLS, NPC> & _src1, float alpha,
xf::Mat<SRC_T, ROWS, COLS, NPC> & _src2, float beta, float gamma,
xf::Mat<SRC_T, ROWS, COLS, NPC> & _dst)
```

Parameter Descriptions

The following table describes the template and the function parameters.

Table 48: **Addweighted Function Parameter Descriptions**

Parameter	Description
SRC_T	Input Pixel Type. 8-bit, unsigned,1 channel is supported (XF_8UC1)
DST_T	Output Pixel Type. 8-bit, unsigned,1 channel is supported (XF_8UC1)
ROWS	Maximum height of input and output image
COLS	Maximum width of input and output image. In case of N-pixel parallelism, width should be multiple of N

Table 48: **Addweighted Function Parameter Descriptions** *(cont'd)*

Parameter	Description
NPC	Number of pixels to be processed per cycle; possible options are XF_NPPC1 and XF_NPPC8 for 1 pixel and 8 pixel operations respectively.
_src1	First Input image
Alpha	Weight applied on first image
_src2	Second Input image
Beta	Weight applied on second image
gamma	Scalar added to each sum
_dst	Output image

Resource Utilization

The following table summarizes the resource utilization of the Addweighted function in Resource optimized (8 pixel) mode and normal mode, as generated in Vivado HLS 2019.1 version tool for the Xczu9eg-ffvb1156-1-i-es1 FPGA.

Table 49: **Addweighted Function Resource Utilization Summary**

Name	Resource Utilization	
	1 pixel per clock operation	8 pixel per clock operation
	300 MHz	150 MHz
BRAM_18K	0	0
DSP48E	11	25
FF	903	680
LUT	851	1077
CLB	187	229

Performance Estimate

The following table summarizes a performance estimate of the kernel in different configurations, generated using Vivado HLS 2019.1 tool for Xczu9eg-ffvb1156-1-i-es1 FPGA to process a grayscale HD (1080x1920) image.

Table 50: **Addweighted Function Performance Estimate Summary**

Operating Mode	Latency Estimate	
	Operating Frequency (MHz)	Latency (ms)
1 pixel	300	6.9
8 pixel	150	1.7

Bilateral Filter

In general, any smoothing filter smoothens the image which will affect the edges of the image. To preserve the edges while smoothing, a bilateral filter can be used. In an analogous way as the Gaussian filter, the bilateral filter also considers the neighboring pixels with weights assigned to each of them. These weights have two components, the first of which is the same weighing used by the Gaussian filter. The second component takes into account the difference in the intensity between the neighboring pixels and the evaluated one.

The bilateral filter applied on an image is:

$$BF[I]_p = \frac{1}{W_p} \sum_{q \epsilon S} G_{\sigma_s}(\| p - q \|) G_{\sigma_r}(\| I_p - I_q \|) I_q$$

Where

$$W_p = \sum_{q \epsilon S} G_{\sigma_s}(\| p - q \|) G_{\sigma_r}(\| I_p - I_q \|)$$

and G_σ is a gaussian filter with variance σ.

The gaussian filter is given by: $G_\sigma = e^{\frac{-(x^2 + y^2)}{2\sigma^2}}$

API Syntax

```
template<int FILTER_SIZE, int BORDER_TYPE, int TYPE, int ROWS, int COLS,
int NPC=1>
void bilateralFilter (
xf::Mat<int TYPE, int ROWS, int COLS, int NPC> src,
xf::Mat<int TYPE, int ROWS, int COLS, int NPC> dst,
float sigma_space, float sigma_color )
```

Parameter Descriptions

The following table describes the template and the function parameters.

Table 51: **bilateralFilter Function Parameter Descriptions**

Parameter	Description
FILTER_SIZE	Filter size. Filter size of 3 (XF_FILTER_3X3), 5 (XF_FILTER_5X5) and 7 (XF_FILTER_7X7) are supported
BORDER_TYPE	Border type supported is XF_BORDER_CONSTANT
TYPE	Input and output pixel type. Only 8-bit, unsigned, 1 channel, and 3 channels are supported (XF_8UC1 and XF_8UC3)
ROWS	Maximum height of input and output image.
COLS	Maximum width of input and output image (must be multiple of 8, for 8-pixel operation)

Table 51: **bilateralFilter Function Parameter Descriptions** *(cont'd)*

Parameter	Description
NPC	Number of pixels to be processed per cycle; this function supports only XF_NPPC1 or 1 pixel per cycle operations.
src	Input image
dst	Output image
sigma_space	Standard deviation of filter in spatial domain
sigma_color	Standard deviation of filter used in color space

Resource Utilization

The following table summarizes the resource utilization of the kernel in different configurations, generated using Vivado HLS 2019.1 version tool for the Xczu9eg-ffvb1156-1-i-es1 FPGA, to progress a grayscale HD (1080x1920) image.

Table 52: **bilateralFilter Resource Utilization Summary**

Operating Mode	Filter Size	Operating Frequency (MHz)	Utilization Estimate			
			BRAM_18K	DSP_48Es	FF	LUT
1 pixel	3x3	300	6	22	4934	4293
	5x5	300	12	30	5481	4943
	7x7	300	37	48	7084	6195

The following table summarizes the resource utilization of the kernel in different configurations, generated using Vivado HLS 2019.1 version tool for the Xczu9eg-ffvb1156-1-i-es1 FPGA, to progress a 4K 3 channel image.

Table 53: **bilateralFilter Resource Utilization Summary**

Operating Mode	Filter Size	Operating Frequency (MHz)	Utilization Estimate			
			BRAM_18K	DSP_48Es	FF	LUT
1 pixel	3x3	300	12	32	8342	7442
	5x5	300	27	57	10663	8857
	7x7	300	49	107	12870	12181

Performance Estimate

The following table summarizes a performance estimate of the kernel in different configurations, as generated using Vivado HLS 2019.1 tool for Xczu9eg-ffvb1156-1-i-es1 FPGA, to process a grayscale HD (1080x1920) image.

Table 54: **bilateralFilter Function Performance Estimate Summary**

Operating Mode	Filter Size	Latency Estimate 300 MHz Max (ms)
1 pixel	3x3	7.18
	5x5	7.20
	7x7	7.22

Deviation from OpenCV

Unlike OpenCV, xfOpenCV only supports filter sizes of 3, 5 and 7.

Bit Depth Conversion

The `convertTo` function converts the input image bit depth to the required bit depth in the output image.

API Syntax

```
template <int SRC_T, int DST_T, int ROWS, int COLS, int NPC=1>
void convertTo(xf::Mat<SRC_T, ROWS, COLS, NPC> &_src_mat, xf::Mat<DST_T,
ROWS, COLS, NPC> &_dst_mat, ap_uint<4> _convert_type, int _shift)
```

Parameter Descriptions

The following table describes the template and the function parameters.

Table 55: **convertTo Function Parameter Descriptions**

Parameter	Description
SRC_T	Input pixel type. 8-bit, unsigned, 1 channel (XF_8UC1), 16-bit, unsigned, 1 channel (XF_16UC1), 16-bit, signed, 1 channel (XF_16SC1), 32-bit, unsigned, 1 channel (XF_32UC1) 32-bit, signed, 1 channel (XF_32SC1) are supported.
DST_T	Output pixel type. 8-bit, unsigned, 1 channel (XF_8UC1), 16-bit, unsigned, 1 channel (XF_16UC1), 16-bit, signed, 1 channel (XF_16SC1), 32-bit, unsigned, 1 channel (XF_32UC1) 32-bit, signed, 1 channel (XF_32SC1) are supported.
ROWS	Height of input and output images
COLS	Width of input and output images
NPC	Number of pixels to be processed per cycle; possible options are XF_NPPC1 and XF_NPPC8 for 1 pixel and 8 pixel operations respectively. XF_NPPC8 is not supported with the 32-bit input and output pixel type.
_src_mat	Input image

Table 55: **convertTo Function Parameter Descriptions** *(cont'd)*

Parameter	Description
_dst_mat	Output image
_convert_type	This parameter specifies the type of conversion required. (See XF_convert_bit_depth_e enumerated type in file `xf_params.h` for possible values.)
_shift	Optional scale factor

Possible Conversions

The following table summarizes supported conversions. The rows are possible input image bit depths and the columns are corresponding possible output image bit depths (U=unsigned, S=signed).

Table 56: **convertTo Function Supported Conversions**

INPUT/OUTPUT	U8	U16	S16	U32	S32
U8	NA	yes	yes	NA	yes
U16	yes	NA	NA	NA	yes
S16	yes	NA	NA	NA	yes
U32	NA	NA	NA	NA	NA
S32	yes	yes	yes	NA	NA

Resource Utilization

The following table summarizes the resource utilization of the convertTo function, generated using Vivado HLS 2019.1 tool for the Xilinx® Xczu9eg-ffvb1156-1-i-es1 FPGA, to process a grayscale HD (1080x1920) image.

Table 57: **convertTo Function Resource Utilization Summary For XF_CONVERT_8U_TO_16S Conversion**

Operating Mode	Operating Frequency (MHz)	Utilization Estimate				
		BRAM_18K	DSP_48Es	FF	LUT	CLB
1 pixel	300	0	8	581	523	119
8 pixel	150	0	8	963	1446	290

Table 58: **convertTo Function Resource Utilization Summary For XF_CONVERT_16U_TO_8U Conversion**

Operating Mode	Operating Frequency (MHz)	Utilization Estimate				
		BRAM_18K	DSP_48Es	FF	LUT	CLB
1 pixel	300	0	8	591	541	124
8 pixel	150	0	8	915	1500	308

Performance Estimate

The following table summarizes the performance in different configurations, as generated using Vivado HLS 2019.1 tool for the Xczu9eg-ffvb1156-1-i-es1, to process a grayscale HD (1080x1920) image.

Table 59: **convertTo Function Performance Estimate Summary**

Operating Mode	Latency Estimate
	Max Latency
1 pixel operation (300 MHz)	6.91 ms
8 pixel operation (150 MHz)	1.69 ms

Bitwise AND

The `bitwise_and` function performs the bitwise AND operation for each pixel between two input images, and returns an output image.

$$I_{out}(x, y) = I_{in1}(x, y) \;\&\; I_{in2}(x, y)$$

Where,

- $I_{out}(x, y)$ is the intensity of output image at (x, y) position
- $I_{in1}(x, y)$ is the intensity of first input image at (x, y) position
- $I_{in2}(x, y)$ is the intensity of second input image at (x, y) position

API Syntax

```
template<int SRC_T, int ROWS, int COLS, int NPC=1>
void bitwise_and (
xf::Mat<int SRC_T, int ROWS, int COLS, int NPC> src1,
xf::Mat<int SRC_T, int ROWS, int COLS, int NPC> src2,
xf::Mat<int SRC_T, int ROWS, int COLS, int NPC> dst )
```

Parameter Descriptions

The following table describes the template and the function parameters.

Table 60: **bitwise_and Function Parameter Descriptions**

Parameter	Description
SRC_T	Input and output pixel type. Supports 1 channel and 3 channels (XF_8UC1 and XF_8UC3)
ROWS	Maximum height of input and output image.
COLS	Maximum width of input and output image (must be a multiple of 8, for 8 pixel mode)
NPC	Number of pixels to be processed per cycle; possible options are XF_NPPC1 and XF_NPPC8 for 1 pixel and 8 pixel operations, respectively.
src1	Input image
src2	Input image
dst	Output image

Resource Utilization

The following table summarizes the resource utilization in different configurations, generated using Vivado HLS 2019.1 tool for the Xczu9eg-ffvb1156-1-i-es1 FPGA, to process a grayscale HD (1080x1920) image.

Table 61: **bitwise_and Function Resource Utilization Summary**

Operating Mode	Operating Frequency (MHz)	Utilization Estimate				
		BRAM_18K	DSP_48Es	FF	LUT	CLB
1 pixel	300	0	0	62	44	10
8 pixel	150	0	0	59	72	13

The following table summarizes the resource utilization in different configurations, generated using Vivado HLS 2019.1 tool for the Xczu9eg-ffvb1156-1-i-es1 FPGA, to process a 4K 3Channel image

Table 62: **bitwise_and Function Resource Utilization Summary**

Operating Mode	Operating Frequency (MHz)	Utilization Estimate				
		BRAM_18K	DSP_48Es	FF	LUT	CLB
1 pixel	300	0	1	155	61	22

Performance Estimate

The following table summarizes the performance in different configurations, as generated using Vivado HLS 2019.1 tool for the Xczu9eg-ffvb1156-1-i-es1, to process a grayscale HD (1080x1920) image.

Table 63: **bitwise_and Function Performance Estimate Summary**

Operating Mode	Latency Estimate
	Max Latency (ms)
1 pixel operation (300 MHz)	6.9
8 pixel operation (150 MHz)	1.7

Bitwise NOT

The `bitwise_not` function performs the pixel wise bitwise NOT operation for the pixels in the input image, and returns an output image. $I_{out}(x, y) = \sim I_{in}(x, y)$

Where,

- $I_{out}(x, y)$ is the intensity of output image at (x, y) position
- $I_{in}(x, y)$ is the intensity of input image at (x, y) position

API Syntax

```
template<int SRC_T, int ROWS, int COLS, int NPC=1>
void bitwise_not (
xf::Mat<int SRC_T, int ROWS, int COLS, int NPC> src,
xf::Mat<int SRC_T, int ROWS, int COLS, int NPC> dst )
```

Parameter Descriptions

The following table describes the template and the function parameters.

Table 64: **bitwise_not Function Parameter Descriptions**

Parameter	Description
SRC_T	Input and output pixel type. Supports 1 channel and 3 channels (XF_8UC1 and XF_8UC3).
ROWS	Maximum height of input and output image.
COLS	Maximum width of input and output image. Must be a multiple of 8 for 8 pixel mode.
NPC	Number of pixels to be processed per cycle; possible options are XF_NPPC1 and XF_NPPC8 for 1 pixel and 8 pixel operations, respectively.
src	Input image
dst	Output image

Resource Utilization

The following table summarizes the resource utilization in different configurations, generated using Vivado HLS 2019.1 tool for the Xczu9eg-ffvb1156-1-i-es1 FPGA, to process a grayscale HD (1080x1920) image.

Table 65: **bitwise_not Function Resource Utilization Summary**

Operating Mode	Operating Frequency (MHz)	Utilization Estimate				
		BRAM_18K	DSP_48Es	FF	LUT	CLB
1 pixel	300	0	0	97	78	20
8 pixel	150	0	0	88	97	21

The following table summarizes the resource utilization in different configurations, generated using Vivado HLS 2019.1 tool for the Xczu9eg-ffvb1156-1-i-es1 FPGA, to process a 4K 3Channel image.

Table 66: **bitwise_not Function Resource Utilization Summary**

Operating Mode	Operating Frequency (MHz)	Utilization Estimate				
		BRAM_18K	DSP_48Es	FF	LUT	CLB
1 pixel	300	0	1	155	61	22

Performance Estimate

The following table summarizes the performance in different configurations, as generated using Vivado HLS 2019.1 tool for the Xczu9eg-ffvb1156-1-i-es1, to process a grayscale HD (1080x1920) image.

Table 67: **bitwise_not Function Performance Estimate Summary**

Operating Mode	Latency Estimate
	Max Latency (ms)
1 pixel operation (300 MHz)	6.9
8 pixel operation (150 MHz)	1.7

Bitwise OR

The `bitwise_or` function performs the pixel wise bitwise OR operation between two input images, and returns an output image. $I_{out}(x, y) = I_{in1}(x, y) \mid I_{in2}(x, y)$

Where,

- $I_{out}(x, y)$ is the intensity of output image at (x, y) position
- $I_{in1}(x, y)$ is the intensity of first input image at (x, y) position
- $I_{in2}(x, y)$ is the intensity of second input image at (x, y) position

API Syntax

```
template<int SRC_T, int ROWS, int COLS, int NPC=1>
void bitwise_or (
xf::Mat<int SRC_T, int ROWS, int COLS, int NPC> src1,
xf::Mat<int SRC_T, int ROWS, int COLS, int NPC> src2,
xf::Mat<int SRC_T, int ROWS, int COLS, int NPC> dst )
```

Parameter Descriptions

The following table describes the template and the function parameters.

Table 68: **bitwise_or Function Parameter Descriptions**

Parameter	Description
SRC_T	Input and output pixel type. Supports 1 channel and 3 channels (XF_8UC1 and XF_8UC3).
ROWS	Maximum height of input and output image.
COLS	Maximum width of input and output image. Must be multiple of 8, for 8 pixel mode.

Table 68: **bitwise_or Function Parameter Descriptions** *(cont'd)*

Parameter	Description
NPC	Number of pixels to be processed per cycle; possible options are XF_NPPC1 and XF_NPPC8 for 1 pixel and 8 pixel operations respectively.
src1	Input image
src2	Input image
dst	Output image

Resource Utilization

The following table summarizes the resource utilization in different configurations, generated using Vivado HLS 2019.1 tool for the Xczu9eg-ffvb1156-1-i-es1 FPGA, to process a grayscale HD (1080x1920) image.

Table 69: **bitwise_or Function Resource Utilization Summary**

Operating Mode	Operating Frequency (MHz)	Utilization Estimate				
		BRAM_18K	DSP_48Es	FF	LUT	CLB
1 pixel	300	0	0	62	44	10
8 pixel	150	0	0	59	72	13

The following table summarizes the resource utilization in different configurations, generated using Vivado HLS 2019.1 tool for the Xczu9eg-ffvb1156-1-i-es1 FPGA, to process a 4K 3Channel image

Table 70: **bitwise_or Function Resource Utilization Summary**

Operating Mode	Operating Frequency (MHz)	Utilization Estimate				
		BRAM_18K	DSP_48Es	FF	LUT	CLB
1 pixel	300	0	1	155	61	22

Performance Estimate

The following table summarizes the performance in different configurations, as generated using Vivado HLS 2019.1 tool for the Xczu9eg-ffvb1156-1-i-es1, to process a grayscale HD (1080x1920) image.

Table 71: **bitwise_or Function Performance Estimate Summary**

Operating Mode	Latency Estimate
	Max Latency (ms)
1 pixel operation (300 MHz)	6.9
8 pixel operation (150 MHz)	1.7

Bitwise XOR

The `bitwise_xor` function performs the pixel wise bitwise XOR operation between two input images, and returns an output image, as shown below:

$$I_{out}(x, y) = I_{in1}(x, y) \oplus I_{in2}(x, y)$$

Where,

- $I_{out}(x, y)$ is the intensity of output image at (x, y) position
- $I_{in1}(x, y)$ is the intensity of first input image at (x, y) position
- $I_{in2}(x, y)$ is the intensity of second input image at (x, y) position

API Syntax

```
template<int SRC_T, int ROWS, int COLS, int NPC=1>
void bitwise_xor(
xf::Mat<int SRC_T, int ROWS, int COLS, int NPC> src1,
xf::Mat<int SRC_T, int ROWS, int COLS, int NPC> src2,
xf::Mat<int SRC_T, int ROWS, int COLS, int NPC> dst )
```

Parameter Descriptions

The following table describes the template and the function parameters.

Table 72: **bitwise_xor Function Parameter Descriptions**

Parameter	Description
SRC_T	Input and output pixel type. Supports 1 channel and 3 channels (XF_8UC1 and XF_8UC3).
ROWS	Maximum height of input and output image.
COLS	Maximum width of input and output image. Must be multiple of 8, for 8 pixel mode.
NPC	Number of pixels to be processed per cycle; possible options are XF_NPPC1 and XF_NPPC8 for 1 pixel and 8 pixel operations respectively.
src1	Input image
src2	Input image

Table 72: **bitwise_xor Function Parameter Descriptions** *(cont'd)*

Parameter	Description
dst	Output image

Resource Utilization

The following table summarizes the resource utilization in different configurations, generated using Vivado HLS 2019.1 tool for the Xczu9eg-ffvb1156-1-i-es1 FPGA, to process a grayscale HD (1080x1920) image:

Table 73: **bitwise_xor Function Resource Utilization Summary**

Operating Mode	Operating Frequency (MHz)	Utilization Estimate				
		BRAM_18K	DSP_48Es	FF	LUT	CLB
1 pixel	300	0	0	62	44	10
8 pixel	150	0	0	59	72	13

The following table summarizes the resource utilization in different configurations, generated using Vivado HLS 2019.1 tool for the Xczu9eg-ffvb1156-1-i-es1 FPGA, to process a 4k Channel image

Table 74: **bitwise_xor Function Resource Utilization Summary**

Operating Mode	Operating Frequency (MHz)	Utilization Estimate				
		BRAM_18K	DSP_48Es	FF	LUT	CLB
1 pixel	300	0	1	155	61	22

Performance Estimate

The following table summarizes the performance in different configurations, as generated using Vivado HLS 2019.1 tool for the Xczu9eg-ffvb1156-1-i-es1, to process a grayscale HD (1080x1920) image:

Table 75: **bitwise_xor Function Performance Estimate Summary**

Operating Mode	Latency Estimate
	Max Latency (ms)
1 pixel operation (300 MHz)	6.9
8 pixel operation (150 MHz)	1.7

Box Filter

The `boxFilter` function performs box filtering on the input image. Box filter acts as a low-pass filter and performs blurring over the image. The `boxFilter` function or the box blur is a spatial domain linear filter in which each pixel in the resulting image has a value equal to the average value of the neighboring pixels in the image.

$$K_{box} = \frac{1}{(ksize*ksize)}\begin{bmatrix} 1 & ... & 1 \\ 1 & ... & 1 \\ 1 & ... & 1 \end{bmatrix}$$

API Syntax

```
template<int BORDER_TYPE,int FILTER_TYPE, int SRC_T, int ROWS, int COLS,int
NPC=1,bool USE_URAM=false>
void boxFilter(xf::Mat<SRC_T, ROWS, COLS, NPC> & _src_mat,xf::Mat<SRC_T,
ROWS, COLS, NPC> & _dst_mat)
```

Parameter Descriptions

The following table describes the template and the function parameters.

Table 76: **boxFilter Function Parameter Descriptions**

Parameter	Description
FILTER_SIZE	Filter size. Filter size of 3(XF_FILTER_3X3), 5(XF_FILTER_5X5) and 7(XF_FILTER_7X7) are supported
BORDER_TYPE	Border Type supported is XF_BORDER_CONSTANT
SRC_T	Input and output pixel type. 8-bit, unsigned, 16-bit unsigned and 16-bit signed, 1 channel is supported (XF_8UC1)
ROWS	Maximum height of input and output image.
COLS	Maximum width of input and output image (must be multiple of 8, for 8-pixel operation)
NPC	Number of pixels to be processed per cycle; possible options are XF_NPPC1 and XF_NPPC8 for 1 pixel and 8 pixel operations respectively.
USE_URAM	Enable to map storage structures to UltraRAM
_src_mat	Input image
_dst_mat	Output image

Resource Utilization

The following table summarizes the resource utilization of the kernel in different configurations, generated using Vivado HLS 2019.1 tool for the Xczu9eg-ffvb1156-1-i-es1 FPGA, to process a grayscale HD (1080x1920) image.

Table 77: **boxFilter Function Resource Utilization Summary**

Operating Mode	Filter Size	Operating Frequency (MHz)	Utilization Estimate				
			BRAM_18K	DSP_48Es	FF	LUT	CLB
1 pixel	3x3	300	3	1	545	519	104
	5x5	300	5	1	876	870	189
	7x7	300	7	1	1539	1506	300
8 pixel	3x3	150	6	8	1002	1368	264
	5x5	150	10	8	1576	3183	611
	7x7	150	14	8	2414	5018	942

The following table summarizes the resource utilization of the kernel in different configurations, generated using the SDx™ 2019.1 tool for the xczu7ev-ffvc1156-2-e FPGA, to process a grayscale 4K (3840x2160) image with UltraRAM enable.

Table 78: **boxFilter Function Resource Utilization Summary with UltraRAM enabled**

Operating Mode	Filter Size	Operating Frequency (MHz)	Utilization Estimate				
			BRAM_18K	URAM	DSP_48Es	FF	LUT
1 pixel	3x3	300	0	1	1	821	521
	5x5	300	0	1	1	1204	855
	7x7	300	0	1	1	2083	1431
8 pixel	3x3	150	0	3	8	1263	1480
	5x5	150	0	5	8	1771	3154
	7x7	150	0	7	8	2700	5411

Performance Estimate

The following table summarizes the performance of the kernel in different configurations, as generated using Vivado HLS 2019.1 tool for the Xczu9eg-ffvb1156-1-i-es1, to process a grayscale HD (1080x1920) image:

Table 79: **boxFilter Function Performance Estimate Summary**

Operating Mode	Operating Frequency (MHz)	Filter Size	Latency Estimate
			Max (ms)
1 pixel	300	3x3	7.2
	300	5x5	7.21
	300	7x7	7.22
8 pixel	150	3x3	1.7
	150	5x5	1.7
	150	7x7	1.7

BoundingBox

The `boundingbox` function highlights the region of interest (ROI) from the input image using below equations.

$$P(X,Y) \leq P(xi, yi) \leq P(X,Y')$$

$$P(X',Y) \leq P(xi, yi) \leq P(X',Y')$$

Where,

- P(xi, yi) - Current pixel location
- P(X,Y) - Top left corner of ROI
- P(X,Y') - Top right corner of ROI
- P(X',Y) - Bottom left corner of ROI
- P(X',Y') - Bottom Right of ROI

API Syntax

```
template<int SRC_T, int ROWS, int COLS, int MAX_BOXES=1, int NPC=1>
void boundingbox(xf::Mat<SRC_T, ROWS, COLS, NPC> & _src_mat, xf::Rect_<int>
*roi , xf::Scalar<4,unsigned char > *color, int num_box)
```

Parameter Descriptions

The following table describes the template and the function parameters.

Table 80: **boundingbox Function Parameter Descriptions**

Parameter	Description
SRC_T	Input pixel Type. Only 8-bit, unsigned, 1 channel and 3 channel is supported (XF_8UC1,XF_8UC3).

Table 80: **boundingbox Function Parameter Descriptions** *(cont'd)*

Parameter	Description
ROWS	Maximum height of input and output image.
COLS	Maximum width of input and output image. Must be multiple of NPC.
MAX_BOXES	Maximum number of boxes, fixed to 5.
NPC	Number of pixels to be processed per cycle, possible options are XF_NPPC1 only.
_src_mat	Input image
roi	ROI is a `xf::Rect` object that consists of the left corner of the rectangle along with the height and width of the rectangle.
color	The `xf::Scalar` object consists of color information for each box (ROI).
num_box	Number of boxes to be detected should be equal or less than *MAX_BOXES*.

Resource Utilization

The following table summarizes the resource utilization in different configurations, generated using Vivado HLS 2019.1 tool for the Xczu9eg-ffvb1156-1-i-es1 FPGA.

Table 81: **boundingbox Function Resource Utilization Summary**

Operating Mode	Operating Frequency (MHz)	Utilization Estimate				
		BRAM_18K	DSP_48Es	FF	LUT	CLB
1 pixel	300	5	4	2521	1649	409

Performance Estimate

The following table summarizes the performance of the kernel in 1-pixel mode as generated using Vivado HLS 2019.1 tool for the Xilinx xczu9eg-ffvb1156-2-i-es2 FPGA to process a grayscale 4K (2160x3840) image for highlighting 3 different boundaries(480x640, 100x200, 300x300).

Table 82: **boundingbox Function Performance Estimate Summary**

Operating Mode	Latency Estimate
	Max Latency (ms)
1 pixel operation (300 MHz)	0.15

xfOpenCV Reference:

The `xf::boundingbox` is complaint with below xfOpenCV function:

```
void rectangle(Mat& img, Rect rec, const Scalar& color, int thickness=1,
int lineType=8, int shift=0 )
```

Canny Edge Detection

The Canny edge detector finds the edges in an image or video frame. It is one of the most popular algorithms for edge detection. Canny algorithm aims to satisfy three main criteria:

1. Low error rate: A good detection of only existent edges.
2. Good localization: The distance between edge pixels detected and real edge pixels have to be minimized.
3. Minimal response: Only one detector response per edge.

In this algorithm, the noise in the image is reduced first by applying a Gaussian mask. The Gaussian mask used here is the average mask of size 3x3. Thereafter, gradients along x and y directions are computed using the Sobel gradient function. The gradients are used to compute the magnitude and phase of the pixels. The phase is quantized and the pixels are binned accordingly. Non-maximal suppression is applied on the pixels to remove the weaker edges.

Edge tracing is applied on the remaining pixels to draw the edges on the image. In this algorithm, the canny up to non-maximal suppression is in one kernel and the edge linking module is in another kernel. After non-maxima suppression, the output is represented as 2-bit per pixel, Where:

- `00` - represents the background
- `01` - represents the weaker edge
- `11` - represents the strong edge

The output is packed as 8-bit (four 2-bit pixels) in 1 pixel per cycle operation and packed as 16-bit (eight 2-bit pixels) in 8 pixel per cycle operation. For the edge linking module, the input is 64-bit, such 32 pixels of 2-bit are packed into a 64-bit. The edge tracing is applied on the pixels and returns the edges in the image.

API Syntax

The API syntax for `Canny` is:

```
template<int FILTER_TYPE,int NORM_TYPE,int SRC_T,int DST_T, int ROWS, int
COLS,int NPC,int NPC1,bool USE_URAM=false>
void Canny(xf::Mat<SRC_T, ROWS, COLS, NPC> & _src_mat,xf::Mat<DST_T, ROWS,
COLS, NPC1> & _dst_mat,unsigned char _lowthreshold,unsigned char
_highthreshold)
```

The API syntax for `EdgeTracing` is:

```
template<int SRC_T, int DST_T, int ROWS, int COLS,int NPC_SRC,int
NPC_DST,bool USE_URAM=false>
voidEdgeTracing(xf::Mat<SRC_T, ROWS, COLS, NPC_SRC> & _src,xf::Mat<DST_T,
ROWS, COLS, NPC_DST> & _dst)
```

Parameter Descriptions

The following table describes the `xf::Canny` template and function parameters:

Table 83: **xf::Canny Function Parameter Descriptions**

Parameter	Description
FILTER_TYPE	The filter window dimensions. The options are 3 and 5.
NORM_TYPE	The type of norm used. The options for norm type are L1NORM and L2NORM.
SRC_T	Input pixel type. Only 8-bit, unsigned, 1 channel is supported (XF_8UC1)
DST_T	Output pixel type. Only XF_2UC1 is supported. The output in case of NPC=XF_NPPC1 is 8-bit and packing four 2-bit pixel values into 8-bit. The output in case of NPC=XF_NPPC8 is 16-bit, 8-bit, 2-bit pixel values are packing into 16-bit.
ROWS	Maximum height of input and output image
COLS	Maximum width of input and output image (must be a multiple of 8, in case of 8 pixel mode)
NPC	Number of pixels to be processed per cycle; possible options are XF_NPPC1 and XF_NPPC8 for 1 pixel and 8 pixel operations respectively. In XF_NPPC, the output image pixels are packed and precision is XF_NPPC4. In XF_NPPC8, output pixels precision is XF_NPPC8.
USE_URAM	Enable to map some storage structures to URAM
_src_mat	Input image
_dst_mat	Output image
_lowthreshold	The lower value of threshold for binary thresholding.
_highthreshold	The higher value of threshold for binary thresholding.

The following table describes the `EdgeTracing` template and function parameters:

Table 84: **EdgeTracing Function Parameter Descriptions**

Parameter	Description
SRC_T	Input pixel type
DST_T	Output pixel type
ROWS	Maximum height of input and output image
COLS	Maximum width of input and output image (must be a multiple of 32)
NPC_SRC	Number of pixels to be processed per cycle. Fixed to XF_NPPC32.
NPC_DST	Number of pixels to be written to destination. Fixed to XF_NPPC8.
USE_URAM	Enable to map storage structures to URAM.
_src	Input image
_dst	Output image

Resource Utilization

The following table summarizes the resource utilization of `xf::Canny` and `EdgeTracing` in different configurations, generated using Vivado HLS 2019.1 tool for the Xczu9eg-ffvb1156-1-i-es1 FPGA, to process a grayscale HD (1080x1920) image for Filter size is 3.

Table 85: **xf::Canny and EdgeTracing Function Resource Utilization Summary**

Name	Resource Utilization					
	1 pixel L1NORM,FS:3 300 MHz	1 pixel L2NORM,FS:3 300 MHz	8 pixel L1NORM,FS:3 150 MHz	8 pixel L2NORM,FS:3 150 MHz	Edge Linking 300 MHz	Edge Linking 150 MHz
BRAM_18K	22	18	36	32	84	84
DSP48E	2	4	16	32	3	3
FF	3027	3507	4899	6208	17600	14356
LUT	2626	3170	6518	9560	15764	14274
CLB	606	708	1264	1871	2955	3241

The following table summarizes the resource utilization of `xf::Canny` and `EdgeTracing` in different configurations, generated using SDx 2019.1 tool for the xczu7ev-ffvc1156-2-e FPGA, to process a grayscale 4K image for Filter size is 3.

Table 86: **xf::Canny and EdgeTracing Function Resource Utilization Summary with UltraRAM Enable**

Name	Resource Utilization					
	1 pixel L1NORM,FS:3 300 MHz	1 pixel L2NORM,FS:3 300 MHz	8 pixel L1NORM,FS:3 150 MHz	8 pixel L2NORM,FS:3 150 MHz	Edge Linking 300 MHz	Edge Linking 150 MHz
BRAM_18K	10	8	3	3	4	4
URAM	1	1	15	13	8	8
DSP48E	2	4	16	32	8	8
FF	3184	3749	5006	7174	5581	7054
LUT	2511	2950	6695	9906	4092	6380

Performance Estimate

The following table summarizes the performance of the kernel in different configurations, as generated using Vivado HLS 2019.1 tool for the Xczu9eg-ffvb1156-1-i-es1, to process a grayscale HD (1080x1920) image for L1NORM, filter size is 3 and including the edge linking module.

Table 87: **xf::Canny and EdgeTracing Function Performance Estimate Summary**

Operating Mode	Latency Estimate	
	Operating Frequency (MHz)	Latency (ms)
1 pixel	300	10.8
8 pixel	150	8.5

Deviation from OpenCV

In OpenCV Canny function, the Gaussian blur is not applied as a pre-processing step.

Channel Combine

The `merge` function, merges single channel images into a multi-channel image. The number of channels to be merged should be four.

API Syntax

```
template<int SRC_T, int DST_T, int ROWS, int COLS, int NPC=1>
void merge(xf::Mat<SRC_T, ROWS, COLS, NPC> &_src1, xf::Mat<SRC_T, ROWS,
COLS, NPC> &_src2, xf::Mat<SRC_T, ROWS, COLS, NPC> &_src3, xf::Mat<SRC_T,
ROWS, COLS, NPC> &_src4, xf::Mat<DST_T, ROWS, COLS, NPC> &_dst)
```

Parameter Descriptions

The following table describes the template and the function parameters.

Table 88: **merge Function Parameter Descriptions**

Parameter	Description
SRC_T	Input pixel type. Only 8-bit, unsigned, 1 channel is supported (XF_8UC1)
DST_T	Output pixel type. Only 8-bit, unsigned, 4 channel is supported (XF_8UC4)
ROWS	Maximum height of input and output image.
COLS	Maximum width of input and output image. Must be multiple of 8 for 8 pixel mode.
NPC	Number of pixels to be processed per cycle; possible options are XF_NPPC1 for 1 pixel operation.
_src1	Input single-channel image
_src2	Input single-channel image
_src3	Input single-channel image
_src4	Input single-channel image
_dst	Output multi-channel image

Resource Utilization

The following table summarizes the resource utilization of the merge function, generated using Vivado HLS 2019.1 tool for the Xczu9eg-ffvb1156-1-i-es1 FPGA, to process 4 single-channel HD (1080x1920) images.

Table 89: **merge Function Resource Utilization Summary**

Operating Mode	Operating Frequency (MHz)	Utilization Estimate				
		BRAM_18K	DSP_48Es	FF	LUT	CLB
1 pixel	300	0	8	494	386	85

Performance Estimate

The following table summarizes the performance in different configurations, as generated using Vivado HLS 2019.1 tool for the Xczu9eg-ffvb1156-1-i-es1, to process 4 single channel HD (1080x1920) images.

Table 90: **merge Function Performance Estimate Summary**

Operating Mode	Latency Estimate
	Max Latency
1 pixel operation (300 MHz)	6.92 ms

Channel Extract

The `extractChannel` function splits a multi-channel array (32-bit pixel-interleaved data) into several single-channel arrays and returns a single channel. The channel to be extracted is specified by using the channel argument.

The value of the channel argument is specified by macros defined in the `xf_channel_extract_e` enumerated data type. The following table summarizes the possible values for the `xf_channel_extract_e` enumerated data type:

Table 91: **xf_channel_extract_e Enumerated Data Type Values**

Channel	Enumerated Type
Unknown	XF_EXTRACT_CH_0
Unknown	XF_EXTRACT_CH_1
Unknown	XF_EXTRACT_CH_2
Unknown	XF_EXTRACT_CH_3
RED	XF_EXTRACT_CH_R
GREEN	XF_EXTRACT_CH_G

Table 91: **xf_channel_extract_e Enumerated Data Type Values** *(cont'd)*

Channel	Enumerated Type
BLUE	XF_EXTRACT_CH_B
ALPHA	XF_EXTRACT_CH_A
LUMA	XF_EXTRACT_CH_Y
Cb/U	XF_EXTRACT_CH_U
Cr/V/Value	XF_EXTRACT_CH_V

API Syntax

```
template<int SRC_T, int DST_T, int ROWS, int COLS, int NPC=1>
void extractChannel(xf::Mat<SRC_T, ROWS, COLS, NPC> & _src_mat,
xf::Mat<DST_T, ROWS, COLS, NPC> & _dst_mat, uint16_t _channel)
```

Parameter Descriptions

The following table describes the template and the function parameters.

Table 92: **extractChannel Function Parameter Descriptions**

Parameter	Description
SRC_T	Input pixel type. Only 8-bit, unsigned, 4channel is supported (XF_8UC4)
DST_T	Output pixel type. Only 8-bit, unsigned, 1 channel is supported (XF_8UC1)
ROWS	Maximum height of input and output image
COLS	Maximum width of input and output image. Must be multiple of 8 for 8 pixel mode
NPC	Number of pixels to be processed per cycle; possible options are XF_NPPC1 for 1 pixel operation.
_src_mat	Input multi-channel image
_dst_mat	Output single channel image
_channel	Channel to be extracted (See xf_channel_extract_e enumerated type in file `xf_params.h` for possible values.)

Resource Utilization

The following table summarizes the resource utilization of the extractChannel function, generated using Vivado HLS 2019.1 tool for the Xczu9eg-ffvb1156-1-i-es1 FPGA, to process a 4 channel HD (1080x1920) image.

Table 93: **extractChannel Function Resource Utilization Summary**

Operating Mode	Operating Frequency (MHz)	Utilization Estimate				
		BRAM_18K	DSP_48Es	FF	LUT	CLB
1 pixel	300	0	8	508	354	96

Performance Estimate

The following table summarizes the performance in different configurations, as generated using Vivado HLS 2019.1 tool for the Xczu9eg-ffvb1156-1-i-es1, to process a 4 channel HD (1080x1920) image.

Table 94: **extractChannel Function Performance Estimate Summary**

Operating Mode	Latency Estimate
	Max Latency (ms)
1 pixel operation (300 MHz)	6.92

Color Conversion

The color conversion functions convert one image format to another image format, for the combinations listed in the following table. The rows represent the input formats and the columns represent the output formats. Supported conversions are discussed in the following sections.

I/O Formats	RGBA	NV12	NV21	IYUV	UYVY	YUYV	YUV4	RGB	BGR
RGBA	N/A	For details, see the RGBA to NV12	For details, see the RGBA to NV21	For details, see the RGBA/RGB to IYUV			For details, see the RGBA/RGB to YUV4		
NV12	For details, see the NV12 to RGBA	N/A	For details, see the NV12 to NV21/ NV21 to NV12	For details, see the NV12 to IYUV	For details, see the NV12/ NV21 to UYVY/ YUYV	For details, see the NV12/ NV21 to UYVY/ YUYV	For details, see the NV12 to YUV4	For details, see the NV12/ NV21 to RGB/ BGR	For details, see the NV12/ NV21 to RGB/ BGR
NV21	For details, see the NV21 to RGBA	For details, see the NV12 to NV21/ NV21 to NV12	N/A	For details, see the NV21 to IYUV	For details, see the NV12/ NV21 to UYVY/ YUYV	For details, see the NV12/ NV21 to UYVY/ YUYV	For details, see the NV21 to YUV4	For details, see the NV12/ NV21 to RGB/ BGR	For details, see the NV12/ NV21 to RGB/ BGR
IYUV	For details, see the IYUV to RGBA/RGB	For details, see the IYUV to NV12		N/A			For details, see the IYUV to YUV4	For details, see the IYUV to RGBA/RGB	
UYVY	For details, see the UYVY to RGBA	For details, see the UYVY to NV12		For details, see the UYVY to IYUV	N/A				
YUYV	For details, see the YUYV to RGBA	For details, see the YUYV to NV12		For details, see the YUYV to IYUV		N/A			

YUV4							N/A		
RGB		For details see theRGB/BGR to NV12/NV21	For details see theRGB/BGR to NV12/NV21	For details see the RGBA/RGB to IYUV	For details see theRGB/BGR to UYVY/YUYV	For details see theRGB/BGR to UYVY/YUYV	For details see the RGBA/RGB to YUV4		For details see theBGR to RGB / RGB to BGR
BGR		For details see theRGB/BGR to NV12/NV21	For details see theRGB/BGR to NV12/NV21		For details see the RGB/BGR to UYVY/YUYV	For details see the RGB/BGR to UYVY/YUYV		For details see theBGR to RGB / RGB to BGR	

Other conversions

Few other conversions are also added. BGR/RGB<->HSV,BGR/RGB<->HLS,BGR/RGB<->YCrCb,BGR/RGB<->XYZ and RGB<->BGR conversions are added.

RGB to YUV Conversion Matrix

Following is the formula to convert RGB data to YUV data:

$$\begin{bmatrix} Y \\ U \\ V \end{bmatrix} = \begin{bmatrix} 0.257 & 0.504 & 0.098 & 16 \\ -0.148 & -0.291 & 0.439 & 128 \\ 0.439 & -0.368 & -0.071 & 128 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \\ 1 \end{bmatrix}$$

YUV to RGB Conversion Matrix

Following is the formula to convert YUV data to RGB data:

$$\begin{bmatrix} R \\ G \\ B \end{bmatrix} = \begin{bmatrix} 1.164 & 0 & 1.596 \\ 1.164 & -0.391 & -0.813 \\ 1.164 & 2.018 & 0 \end{bmatrix} \begin{bmatrix} (Y - 16) \\ (U - 128) \\ (V - 128) \end{bmatrix}$$

Source: http://www.fourcc.org/fccyvrgb.php

RGBA/RGB to YUV4

The `rgba2yuv4` function converts a 4-channel RGBA image to YUV444 format and the `rgb2yuv4` function converts a 3-channel RGB image to YUV444 format. The function outputs Y, U, and V streams separately.

API Syntax

```
template <int SRC_T, int DST_T, int ROWS, int COLS, int NPC=1>
void rgba2yuv4(xf::Mat<SRC_T, ROWS, COLS, NPC> & _src, xf::Mat<DST_T, ROWS,
COLS, NPC> & _y_image, xf::Mat<DST_T, ROWS, COLS, NPC> & _u_image,
xf::Mat<DST_T, ROWS, COLS, NPC> & _v_image)
```

```
template <int SRC_T, int DST_T, int ROWS, int COLS, int NPC=1>
void rgb2yuv4(xf::Mat<SRC_T, ROWS, COLS, NPC> & _src, xf::Mat<DST_T, ROWS,
COLS, NPC> & _y_image, xf::Mat<DST_T, ROWS, COLS, NPC> & _u_image,
xf::Mat<DST_T, ROWS, COLS, NPC> & _v_image)
```

Parameter Descriptions

The following table describes the template and the function parameters.

Table 96: **(rgba/rgb)2yuv4 Function Parameter Descriptions**

Parameter	Description
SRC_T	Input pixel type. Only 8-bit, unsigned, 4(RGBA) and 3(RGB)-channel are supported (XF_8UC4 and XF_8UC3).
DST_T	Output pixel type. Only 8-bit, unsigned, 1-channel is supported (XF_8UC1).
ROWS	Maximum height of input and output image.
COLS	Maximum width of input and output image. Must be a multiple of 8 for 8 pixel mode.
NPC	Number of pixels to be processed per cycle; possible options are XF_NPPC1 and XF_NPPC8 for 1 pixel and 8 pixel operations respectively.
_src	Input Y plane of size (ROWS, COLS).
_y_image	Output Y image of size (ROWS, COLS).
_u_image	Output U image of size (ROWS, COLS).
_v_image	Output V image of size (ROWS, COLS).

Resource Utilization

The following table summarizes the resource utilization of RGBA/RGB to YUV4 for different configurations, generated using Vivado HLS 2019.1 tool for the Xczu9eg-ffvb1156-1-i-es1 FPGA, to process a HD (1080x1920) image.

Table 97: **(rgba/rgb)2yuv4 Function Resource Utilization Summary**

Operating Mode	Operating Frequency (MHz)	Utilization Estimate				
		BRAM_18K	DSP_48Es	FF	LUT	CLB
1 pixel	300	0	9	589	328	96

Performance Estimate

The following table summarizes the performance of RGBA/RGB to YUV4 for different configurations, as generated using the Vivado HLS 2019.1 version for the Xczu9eg-ffvb1156-1-i-es1, to process a grayscale HD (1080x1920) image.

Table 98: **(rgba/rgb)2yuv4 Function Performance Estimate Summary**

Operating Mode	Latency Estimate
	Max Latency (ms)
1 pixel operation (300 MHz)	1.89

RGBA/RGB to IYUV

The `rgba2iyuv` function converts a 4-channel RGBA image to IYUV (4:2:0) format and the `rgb2iyuv` function converts a 3-channel RGB image to IYUV (4:2:0) format. The function outputs Y, U, and V planes separately. IYUV holds subsampled data, Y is sampled for every RGBA/RGB pixel and U,V are sampled once for 2row and 2column(2x2) pixels. U and V planes are of (rows/2)*(columns/2) size, by cascading the consecutive rows into a single row the planes size becomes (rows/4)*columns.

API Syntax

```
template <int SRC_T, int DST_T, int ROWS, int COLS, int NPC=1>
void rgba2iyuv(xf::Mat<SRC_T, ROWS, COLS, NPC> & _src, xf::Mat<DST_T, ROWS,
COLS, NPC> & _y_image, xf::Mat<DST_T, ROWS/4, COLS, NPC> & _u_image,
xf::Mat<DST_T, ROWS/4, COLS, NPC> & _v_image)
```

```
template <int SRC_T, int DST_T, int ROWS, int COLS, int NPC=1>
void rgb2iyuv(xf::Mat<SRC_T, ROWS, COLS, NPC> & _src, xf::Mat<DST_T, ROWS,
COLS, NPC> & _y_image, xf::Mat<DST_T, ROWS/4, COLS, NPC> & _u_image,
xf::Mat<DST_T, ROWS/4, COLS, NPC> & _v_image)
```

Parameter Descriptions

The following table describes the template and the function parameters.

Table 99: **(rgba/rgb)2iyuv Function Parameter Descriptions**

Parameter	Description
SRC_T	Input pixel type. Only 8-bit,unsigned, 4(RGBA) and 3(RGB)-channel are supported (XF_8UC4 and XF_8UC3).
DST_T	Output pixel type. Only 8-bit,unsigned, 1-channel is supported (XF_8UC1).
ROWS	Maximum height of input and output image.
COLS	Maximum width of input and output image. Must be a multiple of 8 for 8 pixel mode.
NPC	Number of pixels to be processed per cycle; possible options are XF_NPPC1 and XF_NPPC8 for 1 pixel and 8 pixel operations respectively.
_src	Input Y plane of size (ROWS, COLS).

Table 99: **(rgba/rgb)2iyuv Function Parameter Descriptions** *(cont'd)*

Parameter	Description
_y_image	Output Y image of size (ROWS, COLS).
_u_image	Output U image of size (ROWS/4, COLS).
_v_image	Output V image of size (ROWS/4, COLS).

Resource Utilization

The following table summarizes the resource utilization of RGBA/RGB to IYUV for different configurations, generated using Vivado HLS 2019.1 tool for the Xczu9eg-ffvb1156-1-i-es1 FPGA, to process a HD (1080x1920) image.

Table 100: **(rgba/rgb)2iyuv Function Resource Utilization Summary**

Operating Mode	Operating Frequency (MHz)	Utilization Estimate				
		BRAM_18K	DSP_48Es	FF	LUT	CLB
1 pixel	300	0	9	816	472	149

Performance Estimate

The following table summarizes the performance of RGBA/RGB to IYUV for different configurations, as generated using Vivado HLS 2019.1 tool for the Xczu9eg-ffvb1156-1-i-es1, to process a grayscale HD (1080x1920) image.

Table 101: **(rgba/rgb)2iyuv Function Performance Estimate Summary**

Operating Mode	Latency Estimate
	Max Latency (ms)
1 pixel operation (300 MHz)	1.8

RGBA to NV12

The `rgba2nv12` function converts a 4-channel RGBA image to NV12 (4:2:0) format. The function outputs Y plane and interleaved UV plane separately. NV12 holds the subsampled data, Y is sampled for every RGBA pixel and U, V are sampled once for 2row and 2columns (2x2) pixels. UV plane is of (rows/2)*(columns/2) size as U and V values are interleaved.

API Syntax

```
template <int SRC_T, int Y_T, int UV_T, int ROWS, int COLS, int NPC=1>
void rgba2nv12(xf::Mat<SRC_T, ROWS, COLS, NPC> & _src, xf::Mat<Y_T, ROWS,
COLS, NPC> & _y, xf::Mat<UV_T, ROWS/2, COLS/2, NPC> & _uv)
```

Parameter Descriptions

The following table describes the template and the function parameters.

Table 102: **rgba2nv12 Function Parameter Descriptions**

Parameter	Description
SRC_T	Input pixel type. Only 8-bit,unsigned, 4-channel is supported (XF_8UC4).
Y_T	Output pixel type. Only 8-bit,unsigned, 1-channel is supported (XF_8UC1).
UV_T	Output pixel type. Only 8-bit,unsigned, 2-channel is supported (XF_8UC2).
ROWS	Maximum height of input and output image.
COLS	Maximum width of input and output image. Must be a multiple of 8 for 8 pixel mode.
NPC	Number of pixels to be processed per cycle; possible options are XF_NPPC1 and XF_NPPC8 for 1 pixel and 8 pixel operations respectively.
_src	Input RGBA image of size (ROWS, COLS).
_y	Output Y image of size (ROWS, COLS).
_uv	Output UV image of size (ROWS/2, COLS/2).

Resource Utilization

The following table summarizes the resource utilization of RGBA to NV12 for different configurations, generated using Vivado HLS 2019.1 tool for the Xczu9eg-ffvb1156-1-i-es1 FPGA, to process a HD (1080x1920) image.

Table 103: **rgba2nv12 Function Resource Utilization Summary**

Operating Mode	Operating Frequency (MHz)	Utilization Estimate				
		BRAM_18K	DSP_48Es	FF	LUT	CLB
1 pixel	300	0	9	802	452	128

Performance Estimate

The following table summarizes the performance of RGBA to NV12 for different configurations, as generated using Vivado HLS 2019.1 tool for the Xczu9eg-ffvb1156-1-i-es1, to process a grayscale HD (1080x1920) image.

Table 104: **rgba2nv12 Function Performance Estimate Summary**

Operating Mode	Latency Estimate
	Max Latency (ms)
1 pixel operation (300 MHz)	1.8

RGBA to NV21

The `rgba2nv21` function converts a 4-channel RGBA image to NV21 (4:2:0) format. The function outputs Y plane and interleaved VU plane separately. NV21 holds subsampled data, Y is sampled for every RGBA pixel and U, V are sampled once for 2 row and 2 columns (2x2) RGBA pixels. UV plane is of (rows/2)*(columns/2) size as V and U values are interleaved.

API Syntax

```
template <int SRC_T, int Y_T, int UV_T, int ROWS, int COLS, int NPC=1>
void rgba2nv21(xf::Mat<SRC_T, ROWS, COLS, NPC> & _src, xf::Mat<Y_T, ROWS,
COLS, NPC> & _y, xf::Mat<UV_T, ROWS/2, COLS/2, NPC> & _uv)
```

Parameter Descriptions

The following table describes the template and the function parameters.

Table 105: **rgba2nv21 Function Parameter Descriptions**

Parameter	Description
SRC_T	Input pixel type. Only 8-bit, unsigned, 4-channel is supported (XF_8UC4).
Y_T	Output pixel type. Only 8-bit, unsigned, 1-channel is supported (XF_8UC1).
UV_T	Output pixel type. Only 8-bit, unsigned, 1-channel is supported (XF_8UC2).
ROWS	Maximum height of input and output image.
COLS	Maximum width of input and output image. Must be a multiple of 8 for 8 pixel mode.
NPC	Number of pixels to be processed per cycle; possible options are XF_NPPC1 and XF_NPPC8 for 1 pixel and 8 pixel operations respectively.
_src	Input RGBA image of size (ROWS, COLS).
_y	Output Y image of size (ROWS, COLS).
_uv	Output UV image of size (ROWS/2, COLS/2).

Resource Utilization

The following table summarizes the resource utilization of RGBA to NV21 for different configurations, generated using Vivado HLS 2019.1 tool for the Xczu9eg-ffvb1156-1-i-es1 FPGA, to process a HD (1080x1920) image.

Table 106: **rgba2nv21 Function Resource Utilization Summary**

Operating Mode	Operating Frequency (MHz)	Utilization Estimate				
		BRAM_18K	DSP_48Es	FF	LUT	CLB
1 pixel	300	0	9	802	453	131

Performance Estimate

The following table summarizes the performance of RGBA to NV21 for different configurations, as generated using Vivado HLS 2019.1 tool for the Xczu9eg-ffvb1156-1-i-es1, to process a grayscale HD (1080x1920) image.

Table 107: **rgba2nv21 Function Performance Estimate Summary**

Operating Mode	Latency Estimate
	Max Latency (ms)
1 pixel operation (300 MHz)	1.89

YUYV to RGBA

The `yuyv2rgba` function converts a single-channel YUYV (YUV 4:2:2) image format to a 4-channel RGBA image. YUYV is a sub-sampled format, a set of YUYV value gives 2 RGBA pixel values. YUYV is represented in 16-bit values where as, RGBA is represented in 32-bit values.

API Syntax

```
template<int SRC_T, int DST_T, int ROWS, int COLS, int NPC=1>
void yuyv2rgba(xf::Mat<SRC_T, ROWS, COLS, NPC> & _src, xf::Mat<DST_T, ROWS,
COLS, NPC> & _dst)
```

Parameter Descriptions

The following table describes the template and the function parameters.

Table 108: **yuyv2rgba Function Parameter Descriptions**

Parameter	Description
SRC_T	Input pixel type. Only 16-bit, unsigned, 1-channel is supported (XF_16UC1).
DST_T	Output pixel type. Only 8-bit, unsigned, 4-channel is supported (XF_8UC4).
ROWS	Maximum height of input and output image.
COLS	Maximum width of input and output image. Must be a multiple of 8 incase of 8 pixel mode.
NPC	Number of pixels to be processed per cycle; possible options are XF_NPPC1 and XF_NPPC8 for 1 pixel and 8 pixel operations respectively.
_src	Input image of size (ROWS, COLS).
_dst	Output image of size (ROWS, COLS).

Resource Utilization

The following table summarizes the resource utilization of YUYV to RGBA for different configurations, generated using Vivado HLS 2019.1 tool for the Xczu9eg-ffvb1156-1-i-es1 FPGA, to process a HD (1080x1920) image.

Table 109: **yuyv2rgba Function Resource Utilization Summary**

Operating Mode	Operating Frequency (MHz)	Utilization Estimate				
		BRAM_18K	DSP_48Es	FF	LUT	CLB
1 pixel	300	0	6	765	705	165

Performance Estimate

The following table summarizes the performance of UYVY to RGBA for different configurations, as generated using Vivado HLS 2019.1 tool for the Xczu9eg-ffvb1156-1-i-es1, to process a grayscale HD (1080x1920) image.

Table 110: **yuyv2rgba Function Performance Estimate Summary**

Operating Mode	Latency Estimate
	Max Latency (ms)
1 pixel operation (300 MHz)	6.9

YUYV to NV12

The `yuyv2nv12` function converts a single-channel YUYV (YUV 4:2:2) image format to NV12 (YUV 4:2:0) format. YUYV is a sub-sampled format, 1 set of YUYV value gives 2 Y values and 1 U and V value each.

API Syntax

```
template<int SRC_T,int Y_T,int UV_T,int ROWS,int COLS,int NPC=1,int
NPC_UV=1>
void yuyv2nv12(xf::Mat<SRC_T, ROWS, COLS, NPC> & _src,xf::Mat<Y_T, ROWS,
COLS, NPC> & _y_image,xf::Mat<UV_T, ROWS/2, COLS/2, NPC_UV> & _uv_image)
```

Parameter Descriptions

The following table describes the template and the function parameters.

Table 111: **yuyv2nv12 Function Parameter Descriptions**

Parameter	Description
SRC_T	Input pixel type. Only 16-bit, unsigned, 1-channel is supported (XF_16UC1).
Y_T	Output pixel type. Only 8-bit, unsigned, 1-channel is supported (XF_8UC1).
UV_T	Output UV image pixel type. Only 8-bit, unsigned, 2-channel is supported (XF_8UC2).
ROWS	Maximum height of input and output image.
COLS	Maximum width of input and output image. Must be a multiple of 8 for 8 pixel mode.

Table 111: **yuyv2nv12 Function Parameter Descriptions** *(cont'd)*

Parameter	Description
NPC	Number of pixels to be processed per cycle; possible options are XF_NPPC1 and XF_NPPC8 for 1 pixel and 8 pixel operations respectively.
NPC_UV	Number of UV image Pixels to be processed per cycle; possible options are XF_NPPC1 and XF_NPPC8 for 1 pixel and 8 pixel operations respectively.
_src	Input image of size (ROWS, COLS).
_y_image	Output Y plane of size (ROWS, COLS).
_uv_image	Output U plane of size (ROWS/2, COLS/2).

Resource Utilization

The following table summarizes the resource utilization of YUYV to NV12 for different configurations, generated using Vivado HLS 2019.1 tool for the Xczu9eg-ffvb1156-1-i-es1 FPGA, to process a HD (1080x1920) image.

Table 112: **yuyv2nv12 Function Resource Utilization Summary**

Operating Mode	Operating Frequency (MHz)	Utilization Estimate				
		BRAM_18K	DSP_48Es	FF	LUT	CLB
1 pixel	300	0	0	831	491	149
8 pixel	150	0	0	1196	632	161

Performance Estimate

The following table summarizes the performance of YUYV to NV12 for different configurations, as generated using Vivado HLS 2019.1 tool for the Xczu9eg-ffvb1156-1-i-es1, to process a grayscale HD (1080x1920) image.

Table 113: **yuyv2nv12 Function Performance Estimate Summary**

Operating Mode	Latency Estimate
	Max Latency (ms)
1 pixel operation (300 MHz)	6.9
8 pixel operation (150 MHz)	1.7

YUYV to IYUV

The `yuyv2iyuv` function converts a single-channel YUYV (YUV 4:2:2) image format to IYUV(4:2:0) format. Outputs of the function are separate Y, U, and V planes. YUYV is a sub-sampled format, 1 set of YUYV value gives 2 Y values and 1 U and V value each. U, V values of the odd rows are dropped as U, V values are sampled once for 2 rows and 2 columns in the IYUV(4:2:0) format.

API Syntax

```
template<int SRC_T, int DST_T, int ROWS, int COLS, int NPC=1>
void yuyv2iyuv(xf::Mat<SRC_T, ROWS, COLS, NPC> & _src, xf::Mat<DST_T, ROWS,
COLS, NPC> & _y_image, xf::Mat<DST_T, ROWS/4, COLS, NPC> & _u_image,
xf::Mat<DST_T, ROWS/4, COLS, NPC> & _v_image)
```

Parameter Descriptions

The following table describes the template and the function parameters.

Table 114: **yuyv2iyuv Function Parameter Descriptions**

Parameter	Description
SRC_T	Input pixel type. Only 16-bit, unsigned,1 channel is supported (XF_16UC1).
DST_T	Output pixel type. Only 8-bit, unsigned, 1 channel is supported (XF_8UC1).
ROWS	Maximum height of input and output image.
COLS	Maximum width of input and output image. Must be a multiple of 8 for 8 pixel modes.
NPC	Number of pixels to be processed per cycle; possible options are XF_NPPC1 and XF_NPPC8 for 1 pixel and 8 pixel operations respectively.
_src	Input image of size (ROWS, COLS).
_y_image	Output Y plane of size (ROWS, COLS).
_u_image	Output U plane of size (ROWS/4, COLS).
_v_image	Output V plane of size (ROWS/4, COLS).

Resource Utilization

The following table summarizes the resource utilization of YUYV to IYUV for different configurations, generated using Vivado HLS 2019.1 tool for the Xilinx Xczu9eg-ffvb1156-1-i-es1 FPGA, to process a HD (1080x1920) image.

Table 115: **yuyv2iyuv Function Resource Utilization Summary**

Operating Mode	Operating Frequency (MHz)	Utilization Estimate				
		BRAM_18K	DSP_48Es	FF	LUT	CLB
1 pixel	300	0	0	835	497	149

Table 115: **yuyv2iyuv Function Resource Utilization Summary** *(cont'd)*

Operating Mode	Operating Frequency (MHz)	Utilization Estimate				
		BRAM_18K	DSP_48Es	FF	LUT	CLB
8 pixel	150	0	0	1428	735	210

Performance Estimate

The following table summarizes the performance of YUYV to IYUV for different configurations, as generated using Vivado HLS 2019.1 tool for the Xilinx Xczu9eg-ffvb1156-1-i-es1, to process a grayscale HD (1080x1920) image.

Table 116: **yuyv2iyuv Function Performance Estimate**

Operating Mode	Latency Estimate
	Max Latency (ms)
1 pixel operation (300 MHz)	6.9
8 pixel operation (150 MHz)	1.7

UYVY to IYUV

The `uyvy2iyuv` function converts a UYVY (YUV 4:2:2) single-channel image to the IYUV format. The outputs of the functions are separate Y, U, and V planes. UYVY is sub sampled format. One set of UYVY value gives two Y values and one U and V value each.

API Syntax

```
template<int SRC_T, int DST_T, int ROWS, int COLS, int NPC=1>
void uyvy2iyuv(xf::Mat<SRC_T, ROWS, COLS, NPC> & _src, xf::Mat<DST_T, ROWS,
COLS, NPC> & _y_image,xf::Mat<DST_T, ROWS/4, COLS, NPC> & _u_image,
xf::Mat<DST_T, ROWS/4, COLS, NPC> & _v_image)
```

Parameter Descriptions

The following table describes the template and the function parameters.

Table 117: **uyvy2iyuv Function Parameter Descriptions**

Parameter	Description
SRC_T	Input pixel type. Only 16-bit, unsigned, 1-channel is supported (XF_16UC1).
DST_T	Output pixel type. Only 8-bit, unsigned, 1-channel is supported (XF_8UC1).
ROWS	Maximum height of input and output image.
COLS	Maximum width of input and output image. Must be a multiple of 8 for 8 pixel mode.

Table 117: **uyvy2iyuv Function Parameter Descriptions** *(cont'd)*

Parameter	Description
NPC	Number of pixels to be processed per cycle; possible options are XF_NPPC1 and XF_NPPC8 for 1 pixel and 8 pixel operations respectively.
_src	Input image of size (ROWS, COLS).
_y_image	Output Y plane of size (ROWS, COLS).
_u_image	Output U plane of size (ROWS/4, COLS).
_v_image	Output V plane of size (ROWS/4, COLS).

Resource Utilization

The following table summarizes the resource utilization of UYVY to IYUV for different configurations, generated using Vivado HLS 2019.1 tool for the Xczu9eg-ffvb1156-1-i-es1 FPGA, to process a HD (1080x1920) image.

Table 118: **uyvy2iyuv Function Resource Utilization Summary**

Operating Mode	Operating Frequency (MHz)	Utilization Estimate				
		BRAM_18K	DSP_48Es	FF	LUT	CLB
1 pixel	300	0	0	835	494	139
8 pixel	150	0	0	1428	740	209

Performance Estimate

The following table summarizes the performance of UYVY to IYUV for different configurations, as generated using Vivado HLS 2019.1 tool for the Xczu9eg-ffvb1156-1-i-es1, to process a grayscale HD (1080x1920) image.

Table 119: **uyvy2iyuv Function Performance Estimate Summary**

Operating Mode	Latency Estimate
	Max Latency (ms)
1 pixel operation (300 MHz)	6.9
8 pixel operation (150 MHz)	1.7

UYVY to RGBA

The `uyvy2rgba` function converts a UYVY (YUV 4:2:2) single-channel image to a 4-channel RGBA image. UYVY is sub sampled format, 1set of UYVY value gives 2 RGBA pixel values. UYVY is represented in 16-bit values where as RGBA is represented in 32-bit values.

API Syntax

```
template<int SRC_T, int DST_T, int ROWS, int COLS, int NPC=1>
void uyvy2rgba(xf::Mat<SRC_T, ROWS, COLS, NPC> & _src, xf::Mat<DST_T, ROWS,
COLS, NPC> & _dst)
```

Parameter Descriptions

The following table describes the template and the function parameters.

Table 120: **uyvy2rgba Function Parameter Descriptions**

Parameter	Description
SRC_T	Input pixel type. Only 16-bit, unsigned, 1-channel is supported (XF_16UC1).
DST_T	Output pixel type. Only 8-bit, unsigned, 1-channel is supported (XF_8UC1).
ROWS	Maximum height of input and output image.
COLS	Maximum width of input and output image. Must be a multiple of 8 for 8 pixel mode.
NPC	Number of pixels to be processed per cycle; possible options are XF_NPPC1 and XF_NPPC8 for 1 pixel and 8 pixel operations respectively.
_src	Input image of size (ROWS, COLS).
_dst	Output image of size (ROWS, COLS).

Resource Utilization

The following table summarizes the resource utilization of UYVY to RGBA for different configurations, generated using Vivado HLS 2019.1 tool for the Xczu9eg-ffvb1156-1-i-es1 FPGA, to process a HD (1080x1920) image.

Table 121: **uyvy2rgba Function Resource Utilization Summary**

Operating Mode	Operating Frequency (MHz)	Utilization Estimate				
		BRAM_18K	DSP_48Es	FF	LUT	CLB
1 pixel	300	0	6	773	704	160

Performance Estimate

The following table summarizes the performance of UYVY to RGBA for different configurations, as generated using Vivado HLS 2019.1 tool for the Xczu9eg-ffvb1156-1-i-es1, to process a grayscale HD (1080x1920) image.

Table 122: **uyvy2rgba Function Performance Estimate Summary**

Operating Mode	Latency Estimate
	Max Latency (ms)
1 pixel operation (300 MHz)	6.8

UYVY to NV12

The `uyvy2nv12` function converts a UYVY (YUV 4:2:2) single-channel image to NV12 format. The outputs are separate Y and UV planes. UYVY is sub sampled format, 1 set of UYVY value gives 2 Y values and 1 U and V value each.

API Syntax

```
template<int SRC_T, int Y_T, int UV_T, int ROWS, int COLS, int NPC=1, int
NPC_UV=1>
void uyvy2nv12(xf::Mat<SRC_T, ROWS, COLS, NPC> & _src,xf::Mat<Y_T, ROWS,
COLS, NPC> & _y_image,xf::Mat<UV_T, ROWS/2, COLS/2, NPC_UV> & _uv_image)
```

Parameter Descriptions

The following table describes the template and the function parameters.

Table 123: **uyvy2nv12 Function Parameter Descriptions**

Parameter	Description
SRC_T	Input pixel type. Only 16-bit, unsigned, 1-channel is supported (XF_16UC1).
Y_T	Output pixel type. Only 8-bit, unsigned, 1-channel is supported (XF_8UC1).
UV_T	Output UV image pixel type. Only 8-bit, unsigned, 2-channel is supported (XF_8UC2).
ROWS	Maximum height of input and output image.
COLS	Maximum width of input and output image. Must be a multiple of 8 for 8 pixel mode.
NPC	Number of pixels to be processed per cycle; possible options are XF_NPPC1 and XF_NPPC8 for 1 pixel and 8 pixel operations respectively.
NPC_UV	Number of UV image Pixels to be processed per cycle; possible options are XF_NPPC1 and XF_NPPC4 for 1 pixel and 8 pixel operations respectively.
_src	Input image of size (ROWS, COLS).
_y_image	Output Y plane of size (ROWS, COLS).
_uv_image	Output U plane of size (ROWS/2, COLS/2).

Resource Utilization

The following table summarizes the resource utilization of UYVY to NV12 for different configurations, generated using Vivado HLS 2019.1 tool for the Xczu9eg-ffvb1156-1-i-es1 FPGA, to process a HD (1080x1920) image.

Table 124: **uyvy2nv12 Function Resource Utilization Summary**

Operating Mode	Operating Frequency (MHz)	Utilization Estimate				
		BRAM_18K	DSP_48Es	FF	LUT	CLB
1 pixel	300	0	0	831	488	131
8 pixel	150	0	0	1235	677	168

Performance Estimate

The following table summarizes the performance of UYVY to NV12 for different configurations, as generated using Vivado HLS 2019.1 tool for the Xczu9eg-ffvb1156-1-i-es1, to process a grayscale HD (1080x1920) image.

Table 125: **uyvy2nv12 Function Performance Estimate Summary**

Operating Mode	Latency Estimate
	Max Latency (ms)
1 pixel operation (300 MHz)	6.9
8 pixel operation (150 MHz)	1.7

IYUV to RGBA/RGB

The `iyuv2rgba` function converts single channel IYUV (YUV 4:2:0) image to a 4-channel RGBA image and `iyuv2rgb` function converts single channel IYUV (YUV 4:2:0) image to a 3-channel RGB image . The inputs to the function are separate Y, U, and V planes. IYUV is sub sampled format, U and V values are sampled once for 2 rows and 2 columns of the RGBA/RGB pixels. The data of the consecutive rows of size (columns/2) is combined to form a single row of size (columns).

API Syntax

```
template<int SRC_T, int DST_T, int ROWS, int COLS, int NPC=1>
void iyuv2rgba(xf::Mat<SRC_T, ROWS, COLS, NPC> & src_y, xf::Mat<SRC_T,
ROWS/4, COLS, NPC> & src_u,xf::Mat<SRC_T, ROWS/4, COLS, NPC> & src_v,
xf::Mat<DST_T, ROWS, COLS, NPC> & _dst0)
```

```
template<int SRC_T, int DST_T, int ROWS, int COLS, int NPC=1>
void iyuv2rgb(xf::Mat<SRC_T, ROWS, COLS, NPC> & src_y, xf::Mat<SRC_T,
ROWS/4, COLS, NPC> & src_u,xf::Mat<SRC_T, ROWS/4, COLS, NPC> & src_v,
xf::Mat<DST_T, ROWS, COLS, NPC> & _dst0)
```

Parameter Descriptions

The following table describes the template and the function parameters.

Table 126: **iyuv2(rgba/rgb) Function Parameter Descriptions**

Parameter	Description
SRC_T	Input pixel type. Only 8-bit, unsigned, 1-channel is supported (XF_8UC1).
DST_T	Output pixel type. Only 8-bit, unsigned, 4(RGBA) and 3(RGB)-channel are supported (XF_8UC4 and XF_8UC3).
ROWS	Maximum height of input and output image.
COLS	Maximum width of input and output image. Must be a multiple of 8 for 8 pixel mode.
NPC	Number of pixels to be processed per cycle; possible options are XF_NPPC1 and XF_NPPC8 for 1 pixel and 8 pixel operations respectively.
src_y	Input Y plane of size (ROWS, COLS).
src_u	Input U plane of size (ROWS/4, COLS).
src_v	Input V plane of size (ROWS/4, COLS).
_dst0	Output RGBA image of size (ROWS, COLS).

Resource Utilization

The following table summarizes the resource utilization of IYUV to RGBA/RGB for different configurations, generated using Vivado HLS 2019.1 tool for the Xczu9eg-ffvb1156-1-i-es1 FPGA, to process a HD (1080x1920) image.

Table 127: **iyuv2(rgba/rgb) Function Resource Utilization Summary**

Operating Mode	Operating Frequency (MHz)	Utilization Estimate				
		BRAM_18K	DSP_48Es	FF	LUT	CLB
1 pixel	300	2	5	1208	728	196

Performance Estimate

The following table summarizes the performance of IYUV to RGBA/RGB for different configurations, as generated using the Vivado HLS 2019.1 version tool for the Xczu9eg-ffvb1156-1-i-es1, to process a grayscale HD (1080x1920) image.

Table 128: **iyuv2(rgba/rgb) Function Performance Estimate Summary**

Operating Mode	Latency Estimate
	Max Latency (ms)
1 pixel operation (300 MHz)	6.9

IYUV to NV12

The `iyuv2nv12` function converts single channel IYUV image to NV12 format. The inputs are separate U and V planes. There is no need of processing Y plane as both the formats have a same Y plane. U and V values are rearranged from plane interleaved to pixel interleaved.

API Syntax

```
template<int SRC_T, int UV_T, int ROWS, int COLS, int NPC =1, int NPC_UV=1>
void iyuv2nv12(xf::Mat<SRC_T, ROWS, COLS, NPC> & src_y, xf::Mat<SRC_T,
ROWS/4, COLS, NPC> & src_u,xf::Mat<SRC_T, ROWS/4, COLS, NPC> &
src_v,xf::Mat<SRC_T, ROWS, COLS, NPC> & _y_image, xf::Mat<UV_T, ROWS/2,
COLS/2, NPC_UV> & _uv_image)
```

Parameter Descriptions

The following table describes the template and the function parameters.

Table 129: **iyuv2nv12 Function Parameter Descriptions**

Parameter	Description
SRC_T	Input pixel type. Only 8-bit, unsigned, 1-channel is supported (XF_8UC1).
UV_T	Output pixel type. Only 8-bit, unsigned, 2-channel is supported (XF_8UC2).
ROWS	Maximum height of input and output image.
COLS	Maximum width of input and output image. Must be a multiple of 8 for 8 pixel mode.
NPC	Number of pixels to be processed per cycle; possible options are XF_NPPC1 and XF_NPPC8 for 1 pixel and 8 pixel operations respectively.
NPC_UV	Number of UV Pixels to be processed per cycle; possible options are XF_NPPC1 and XF_NPPC4 for 1 pixel and 4-pixel operations respectively.
src_y	Input Y plane of size (ROWS, COLS).
src_u	Input U plane of size (ROWS/4, COLS).
src_v	Input V plane of size (ROWS/4, COLS).
_y_image	Output V plane of size (ROWS, COLS).
_uv_image	Output UV plane of size (ROWS/2, COLS/2).

Resource Utilization

The following table summarizes the resource utilization of IYUV to NV12 for different configurations, generated using Vivado HLS 2019.1 tool for the Xczu9eg-ffvb1156-1-i-es1 FPGA, to process a HD (1080x1920) image..

Table 130: **iyuv2nv12 Function Resource Utilization Summary**

Operating Mode	Operating Frequency (MHz)	Utilization Estimate				
		BRAM_18K	DSP_48Es	FF	LUT	CLB
1 pixel	300	0	12	907	677	158
8 pixel	150	0	12	1591	1022	235

Performance Estimate

The following table summarizes the performance of IYUV to NV12 for different configurations, as generated using the Vivado HLS 2019.1 version tool for the Xczu9eg-ffvb1156-1-i-es1, to process a grayscale HD (1080x1920) image.

Table 131: **iyuv2nv12 Function Performance Estimate Summary**

Operating Mode	Latency Estimate
	Max Latency (ms)
1 pixel operation (300 MHz)	6.9
8 pixel operation (150 MHz)	1.7

IYUV to YUV4

The `iyuv2yuv4` function converts a single channel IYUV image to a YUV444 format. Y plane is same for both the formats. The inputs are separate U and V planes of IYUV image and the outputs are separate U and V planes of YUV4 image. IYUV stores subsampled U,V values. YUV format stores U and V values for every pixel. The same U, V values are duplicated for 2 rows and 2 columns (2x2) pixels in order to get the required data in the YUV444 format.

API Syntax

```
template<int SRC_T, int ROWS, int COLS, int NPC=1>
void iyuv2yuv4(xf::Mat<SRC_T, ROWS, COLS, NPC> & src_y, xf::Mat<SRC_T,
ROWS/4, COLS, NPC> & src_u,xf::Mat<SRC_T, ROWS/4, COLS, NPC> &
src_v,xf::Mat<SRC_T, ROWS, COLS, NPC> & _y_image, xf::Mat<SRC_T, ROWS,
COLS, NPC> & _u_image, xf::Mat<SRC_T, ROWS, COLS, NPC> & _v_image)
```

Parameter Descriptions

The following table describes the template and the function parameters.

Table 132: **iyuv2yuv4 Function Parameter Descriptions**

Parameter	Description
SRC_T	Input pixel type. Only 8-bit, unsigned, 1-channel is supported (XF_8UC1).

Table 132: **iyuv2yuv4 Function Parameter Descriptions** *(cont'd)*

Parameter	Description
ROWS	Maximum height of input and output image.
COLS	Maximum width of input and output image. Must be a multiple of 8, for 8 pixel mode.
NPC	Number of pixels to be processed per cycle; possible options are XF_NPPC1 and XF_NPPC8 for 1 pixel and 8 pixel operations respectively.
src_y	Input Y plane of size (ROWS, COLS).
src_u	Input U plane of size (ROWS/4, COLS).
src_v	Input V plane of size (ROWS/4, COLS).
_y_image	Output Y image of size (ROWS, COLS).
_u_image	Output U image of size (ROWS, COLS).
_v_image	Output V image of size (ROWS, COLS).

Resource Utilization

The following table summarizes the resource utilization of IYUV to YUV4 for different configurations, generated using Vivado HLS 2019.1 tool for the Xczu9eg-ffvb1156-1-i-es1 FPGA, to process a HD (1080x1920) image.

Table 133: **iyuv2yuv4 Function Resource Utilization Summary**

Operating Mode	Operating Frequency (MHz)	Utilization Estimate				
		BRAM_18K	DSP_48Es	FF	LUT	CLB
1 pixel	300	0	0	1398	870	232
8 pixel	150	0	0	2134	1214	304

Performance Estimate

The following table summarizes the performance of IYUV to YUV4 for different configurations, as generated using the Vivado HLS 2019.1 version tool for the Xczu9eg-ffvb1156-1-i-es1, to process a grayscale HD (1080x1920) image.

Table 134: **iyuv2yuv4 Function Performance Estimate Summary**

Operating Mode	Latency Estimate
	Max Latency (ms)
1 pixel operation (300 MHz)	13.8
8 pixel operation (150 MHz)	3.4

NV12 to IYUV

The `nv122iyuv` function converts NV12 format to IYUV format. The function inputs the interleaved UV plane and the outputs are separate U and V planes. There is no need of processing the Y plane as both the formats have a same Y plane. U and V values are rearranged from pixel interleaved to plane interleaved.

API Syntax

```
template<int SRC_T, int UV_T, int ROWS, int COLS, int NPC=1, int NPC_UV=1>
void nv122iyuv(xf::Mat<SRC_T, ROWS, COLS, NPC> & src_y, xf::Mat<UV_T,
ROWS/2, COLS/2, NPC_UV> & src_uv,xf::Mat<SRC_T, ROWS, COLS, NPC> &
_y_image,xf::Mat<SRC_T, ROWS/4, COLS, NPC> & _u_image,xf::Mat<SRC_T,
ROWS/4, COLS, NPC> & _v_image)
```

Parameter Descriptions

The following table describes the template and the function parameters.

Table 135: **nv122iyuv Function Parameter Descriptions**

Parameter	Description
SRC_T	Input pixel type. Only 8-bit, unsigned, 1-channel is supported (XF_8UC1).
UV_T	Input pixel type. Only 8-bit, unsigned, 2-channel is supported (XF_8UC2).
ROWS	Maximum height of input and output image.
COLS	Maximum width of input and output image (must be a multiple of 8, for 8 pixel mode).
NPC	Number of pixels to be processed per cycle; possible options are XF_NPPC1 and XF_NPPC8 for 1 pixel and 8 pixel operations respectively.
NPC_UV	Number of UV image Pixels to be processed per cycle; possible options are XF_NPPC1 and XF_NPPC4 for 1 pixel and 4-pixel operations respectively.
src_y	Input Y plane of size (ROWS, COLS).
src_uv	Input UV plane of size (ROWS/2, COLS/2).
_y_image	Output Y plane of size (ROWS, COLS).
_u_image	Output U plane of size (ROWS/4, COLS).
_v_image	Output V plane of size (ROWS/4, COLS).

Resource Utilization

The following table summarizes the resource utilization of NV12 to IYUV for different configurations, as generated in the Vivado HLS 2019.1 version tool for the Xczu9eg-ffvb1156-1-i-es1 FPGA, to process a HD (1080x1920) image.

Table 136: **nv122iyuv Function Resource Utilization Summary**

Operating Mode	Operating Frequency (MHz)	Utilization Estimate				
		BRAM_18K	DSP_48Es	FF	LUT	CLB
1 pixel	300	0	1	1344	717	208
8 pixel	150	0	1	1961	1000	263

Performance Estimate

The following table summarizes the performance of NV12 to IYUV for different configurations, as generated using the Vivado HLS 2019.1 version tool for the Xczu9eg-ffvb1156-1-i-es1, to process a grayscale HD (1080x1920) image.

Table 137: **nv122iyuv Function Performance Estimate Summary**

Operating Mode	Latency Estimate
	Max Latency (ms)
1 pixel operation (300 MHz)	6.9
8 pixel operation (150 MHz)	1.7

NV12 to RGBA

The `nv122rgba` function converts NV12 image format to a 4-channel RGBA image. The inputs to the function are separate Y and UV planes. NV12 holds sub sampled data, Y plane is sampled at unit rate and 1 U and 1 V value each for every 2x2 Y values. To generate the RGBA data, each U and V value is duplicated (2x2) times.

API Syntax

```
template<int SRC_T, int UV_T, int DST_T, int ROWS, int COLS, int NPC=1>
void nv122rgba(xf::Mat<SRC_T, ROWS, COLS, NPC> & src_y,xf::Mat<UV_T,
ROWS/2, COLS/2, NPC> & src_uv,xf::Mat<DST_T, ROWS, COLS, NPC> & _dst0)
```

Parameter Descriptions

The following table describes the template and the function parameters.

Table 138: **nv122rgba Function Parameter Descriptions**

Parameter	Description
SRC_T	Input pixel type. Only 8-bit, unsigned, 1-channel is supported (XF_8UC1).
UV_T	Input pixel type. Only 8-bit, unsigned, 2-channel is supported (XF_8UC2).
DST_T	Output pixel type. Only 8-bit,unsigned,4channel is supported (XF_8UC4).

Table 138: **nv122rgba Function Parameter Descriptions** *(cont'd)*

Parameter	Description
ROWS	Maximum height of input and output image.
COLS	Maximum width of input and output image. Must be a multiple of 8, for 8 pixel mode.
NPC	Number of pixels to be processed per cycle; possible options are XF_NPPC1 and XF_NPPC8 for 1 pixel and 8 pixel operations respectively.
src_y	Input Y plane of size (ROWS, COLS).
src_uv	Input UV plane of size (ROWS/2, COLS/2).
_dst0	Output RGBA image of size (ROWS, COLS).

Resource Utilization

The following table summarizes the resource utilization of NV12 to RGBA for different configurations, as generated in the Vivado HLS 2019.1 version tool for the Xczu9eg-ffvb1156-1-i-es1 FPGA, to process a HD (1080x1920) image.

Table 139: **nv122rgba Function Resource Utilization Summary**

Operating Mode	Operating Frequency (MHz)	Utilization Estimate				
		BRAM_18K	DSP_48Es	FF	LUT	CLB
1 pixel	300	2	5	1191	708	195

Performance Estimate

The following table summarizes the performance of NV12 to RGBA for different configurations, as generated using the Vivado HLS 2019.1 version tool for the Xczu9eg-ffvb1156-1-i-es1, to process a grayscale HD (1080x1920) image.

Table 140: **nv122rgba Function Performance Estimate Summary**

Operating Mode	Latency Estimate
	Max Latency (ms)
1 pixel operation (300 MHz)	6.9

NV12 to YUV4

The `nv122yuv4` function converts a NV12 image format to a YUV444 format. The function outputs separate U and V planes. Y plane is same for both the image formats. The UV planes are duplicated 2x2 times to represent one U plane and V plane of the YUV444 image format.

API Syntax

```
template<int SRC_T,int UV_T, int ROWS, int COLS, int NPC=1, int NPC_UV=1>
void nv122yuv4(xf::Mat<SRC_T, ROWS, COLS, NPC> & src_y, xf::Mat<UV_T,
ROWS/2, COLS/2, NPC_UV> & src_uv,xf::Mat<SRC_T, ROWS, COLS, NPC> &
_y_image, xf::Mat<SRC_T, ROWS, COLS, NPC> & _u_image,xf::Mat<SRC_T, ROWS,
COLS, NPC> & _v_image)
```

Parameter Descriptions

The following table describes the template and the function parameters.

Table 141: **nv122yuv4 Function Parameter Descriptions**

Parameter	Description
SRC_T	Input pixel type. Only 8-bit, unsigned, 1-channel is supported (XF_8UC1).
UV_T	Input pixel type. Only 8-bit, unsigned, 2-channel is supported (XF_8UC2).
ROWS	Maximum height of input and output image.
COLS	Maximum width of input and output image (must be a multiple of 8, for 8 pixel mode).
NPC	Number of pixels to be processed per cycle; possible options are XF_NPPC1 and XF_NPPC8 for 1 pixel and 8 pixel operations respectively.
NPC_UV	Number of UV image Pixels to be processed per cycle; possible options are XF_NPPC1 and XF_NPPC4 for 1 pixel and 4-pixel operations respectively.
src_y	Input Y plane of size (ROWS, COLS).
src_uv	Input UV plane of size (ROWS/2, COLS/2).
_y_image	Output Y plane of size (ROWS, COLS).
_u_image	Output U plane of size (ROWS, COLS).
_v_image	Output V plane of size (ROWS, COLS).

Resource Utilization

The following table summarizes the resource utilization of NV12 to YUV4 for different configurations, as generated in the Vivado HLS 2019.1 version tool for the Xczu9eg-ffvb1156-1-i-es1 FPGA, to process a HD (1080x1920) image.

Table 142: **nv122yuv4 Function Resource Utilization Summary**

Operating Mode	Operating Frequency (MHz)	Utilization Estimate				
		BRAM_18K	DSP_48Es	FF	LUT	CLB
1 pixel	300	0	0	1383	832	230
8 pixel	150	0	0	1772	1034	259

Performance Estimate

The following table summarizes the performance of NV12 to YUV4 for different configurations, as generated using the Vivado HLS 2019.1 version tool for the Xczu9eg-ffvb1156-1-i-es1, to process a grayscale HD (1080x1920) image.

Table 143: **nv122yuv4 Function Performance Estimate Summary**

Operating Mode	Latency Estimate
	Max Latency (ms)
1 pixel operation (300 MHz)	13.8
8 pixel operation (150 MHz)	3.4

NV21 to IYUV

The `nv212iyuv` function converts a NV21 image format to an IYUV image format. The input to the function is the interleaved VU plane only and the outputs are separate U and V planes. There is no need of processing Y plane as both the formats have same the Y plane. U and V values are rearranged from pixel interleaved to plane interleaved.

API Syntax

```
template<int SRC_T, int UV_T, int ROWS, int COLS, int NPC=1,int NPC_UV=1>
void nv212iyuv(xf::Mat<SRC_T, ROWS, COLS, NPC> & src_y, xf::Mat<UV_T,
ROWS/2, COLS/2, NPC_UV> & src_uv,xf::Mat<SRC_T, ROWS, COLS, NPC> &
_y_image, xf::Mat<SRC_T, ROWS/4, COLS, NPC> & _u_image,xf::Mat<SRC_T,
ROWS/4, COLS, NPC> & _v_image)
```

Parameter Descriptions

The following table describes the template and the function parameters.

Table 144: **nv212iyuv Function Parameter Descriptions**

Parameter	Description
SRC_T	Input pixel type. Only 8-bit, unsigned, 1-channel is supported (XF_8UC1).
UV_T	Input pixel type. Only 8-bit, unsigned, 2-channel is supported (XF_8UC2).
ROWS	Maximum height of input and output image .
COLS	Maximum width of input and output image. Must be a multiple of 8, for 8 pixel mode.
NPC	Number of pixels to be processed per cycle; possible options are XF_NPPC1 and XF_NPPC8 for 1 pixel and 8 pixel operations respectively.
NPC_UV	Number of UV image Pixels to be processed per cycle; possible options are XF_NPPC1 and XF_NPPC4 for 1 pixel and 4-pixel operations respectively.
src_y	Input Y plane of size (ROWS, COLS).
src_uv	Input UV plane of size (ROWS/2, COLS/2).
_y_image	Output Y plane of size (ROWS, COLS).
_u_image	Output U plane of size (ROWS/4, COLS).

Table 144: **nv212iyuv Function Parameter Descriptions** *(cont'd)*

Parameter	Description
_v_image	Output V plane of size (ROWS/4, COLS).

Resource Utilization

The following table summarizes the resource utilization of NV21 to IYUV for different configurations, as generated in the Vivado HLS 2019.1 version tool for the Xczu9eg-ffvb1156-1-i-es1 FPGA, to process a HD (1080x1920) image.

Table 145: **nv212iyuv Function Resource Utilization Summary**

Operating Mode	Operating Frequency (MHz)	Utilization Estimate				
		BRAM_18K	DSP_48Es	FF	LUT	CLB
1 pixel	300	0	1	1377	730	219
8 pixel	150	0	1	1975	1012	279

Performance Estimate

The following table summarizes the performance of NV21 to IYUV for different configurations, as generated using the Vivado HLS 2019.1 version tool for the Xczu9eg-ffvb1156-1-i-es1, to process a grayscale HD (1080x1920) image.

Table 146: **nv212iyuv Function Performance Estimate Summary**

Operating Mode	Latency Estimate
	Max Latency (ms)
1 pixel operation (300 MHz)	6.9
8 pixel operation (150 MHz)	1.7

NV21 to RGBA

The `nv212rgba` function converts a NV21 image format to a 4-channel RGBA image. The inputs to the function are separate Y and VU planes. NV21 holds sub sampled data, Y plane is sampled at unit rate and one U and one V value each for every 2x2 Yvalues. To generate the RGBA data, each U and V value is duplicated (2x2) times.

API Syntax

```
template<int SRC_T, int UV_T, int DST_T, int ROWS, int COLS, int NPC=1>
void nv212rgba(xf::Mat<SRC_T, ROWS, COLS, NPC> & src_y, xf::Mat<UV_T,
ROWS/2, COLS/2, NPC> & src_uv,xf::Mat<DST_T, ROWS, COLS, NPC> & _dst0)
```

Parameter Descriptions

The following table describes the template and the function parameters.

Table 147: **nv212rgba Function Parameter Descriptions**

Parameter	Description
SRC_T	Input pixel type. Only 8-bit, unsigned, 1-channel is supported (XF_8UC1).
UV_T	Input pixel type. Only 8-bit, unsigned, 2-channel is supported (XF_8UC2).
DST_T	Output pixel type. Only 8-bit, unsigned, 4-channel is supported (XF_8UC4).
ROWS	Maximum height of input and output image.
COLS	Maximum width of input and output image. Must be a multiple of 8, incase of 8 pixel mode.
NPC	Number of pixels to be processed per cycle; possible options are XF_NPPC1 and XF_NPPC8 for 1 pixel and 8 pixel operations respectively.
src_y	Input Y plane of size (ROWS, COLS).
src_uv	Input UV plane of size (ROWS/2, COLS/2).
_dst0	Output RGBA image of size (ROWS, COLS).

Resource Utilization

The following table summarizes the resource utilization of NV21 to RGBA for different configurations, as generated in the Vivado HLS 2019.1 version tool for the Xczu9eg-ffvb1156-1-i-es1 FPGA, to process a HD (1080x1920) image.

Table 148: **nv212rgba Function Resource Utilization Summary**

Operating Mode	Operating Frequency (MHz)	Utilization Estimate				
		BRAM_18K	DSP_48Es	FF	LUT	CLB
1 pixel	300	2	5	1170	673	183

Performance Estimate

The following table summarizes the performance of NV12 to RGBA for different configurations, as generated using the Vivado HLS 2019.1 version tool for the Xczu9eg-ffvb1156-1-i-es1, to process a grayscale HD (1080x1920) image.

Table 149: **nv212rgba Function Performance Estimate Summary**

Operating Mode	Latency Estimate
	Max Latency (ms)
1 pixel operation (300 MHz)	6.9

NV21 to YUV4

The `nv212yuv4` function converts an image in the NV21 format to a YUV444 format. The function outputs separate U and V planes. Y plane is same for both formats. The UV planes are duplicated 2x2 times to represent one U plane and V plane of YUV444 format.

API Syntax

```
template<int SRC_T, int UV_T, int ROWS, int COLS, int NPC=1,int NPC_UV=1>
void nv212yuv4(xf::Mat<SRC_T, ROWS, COLS, NPC> & src_y, xf::Mat<UV_T,
ROWS/2, COLS/2, NPC_UV> & src_uv, xf::Mat<SRC_T, ROWS, COLS, NPC> &
_y_image, xf::Mat<SRC_T, ROWS, COLS, NPC> & _u_image, xf::Mat<SRC_T, ROWS,
COLS, NPC> & _v_image)
```

Parameter Descriptions

The following table describes the template and the function parameters.

Table 150: **nv212yuv4 Function Parameter Descriptions**

Parameter	Description
SRC_T	Input pixel type. Only 8-bit, unsigned, 1-channel is supported (XF_8UC1).
UV_T	Input pixel type. Only 8-bit, unsigned, 2-channel is supported (XF_8UC2).
ROWS	Maximum height of input and output image.
COLS	Maximum width of input and output image (must be a multiple of 8, for 8 pixel mode).
NPC	Number of pixels to be processed per cycle; possible options are XF_NPPC1 and XF_NPPC8 for 1 pixel and 8 pixel operations respectively.
NPC_UV	Number of UV image Pixels to be processed per cycle; possible options are XF_NPPC1 and XF_NPPC4 for 1 pixel and 4-pixel operations respectively.
src_y	Input Y plane of size (ROWS, COLS).
src_uv	Input UV plane of size (ROWS/2, COLS/2).
_y_image	Output Y plane of size (ROWS, COLS).
_u_image	Output U plane of size (ROWS, COLS).
_v_image	Output V plane of size (ROWS, COLS).

Resource Utilization

The following table summarizes the resource utilization of NV21 to YUV4 for different configurations, as generated in the Vivado HLS 2019.1 version tool for the Xczu9eg-ffvb1156-1-i-es1 FPGA, to process a HD (1080x1920) image.

Table 151: **nv212yuv4 Function Resource Utilization Summary**

Operating Mode	Operating Frequency (MHz)	Utilization Estimate				
		BRAM_18K	DSP_48Es	FF	LUT	CLB
1 pixel	300	0	0	1383	817	233
8 pixel	150	0	0	1887	1087	287

Performance Estimate

The following table summarizes the performance of NV21 to YUV4 for different configurations, as generated using the Vivado HLS 2019.1 version tool for the Xczu9eg-ffvb1156-1-i-es1, to process a grayscale HD (1080x1920) image.

Table 152: **nv212yuv4 Function Performance Estimate Summary**

Operating Mode	Latency Estimate
	Max Latency (ms)
1 pixel operation (300 MHz)	13.8
8 pixel operation (150 MHz)	3.5

RGB to GRAY

The `rgb2gray` function converts a 3-channel RGB image to GRAY format.

$$Y= 0.299*R+0.587*G+0.114*B$$

Where,

- Y = Gray pixel
- R= Red channel
- G= Green channel
- B= Blue channel

API Syntax

```
template<int SRC_T, int DST_T, int ROWS, int COLS, int NPC=1>
void rgb2gray(xf::Mat<SRC_T, ROWS, COLS, NPC> & _src, xf::Mat<DST_T, ROWS,
COLS, NPC> & _dst)
```

Parameter Descriptions

The following table describes the template and the function parameters.

Table 153: **RGB2GRAY Function Parameter Descriptions**

Parameter	Description
SRC_T	Input pixel type. Only 8-bit, unsigned, 3-channel is supported (XF_8UC3).
DST_T	Output pixel type. Only 8-bit, unsigned, 1-channel is supported (XF_8UC1)
ROWS	Maximum height of input and output image.
COLS	Maximum width of input and output image.
NPC	Number of pixels to be processed per cycle.
_src	RGB input image
_dst	GRAY output image

Resource Utilization

The following table summarizes the resource utilization of RGB to GRAY for different configurations, as generated in the Vivado HLS 2019.1 version tool for the Xczu9eg-ffvb1156-1-i-es1 FPGA, to process a HD (1080x1920) image.

Table 154: **RGB2GRAY Function Resource Utilization Summary**

Operating Mode	Operating Frequency (MHz)	Utilization Estimate			
		BRAM_18K	DSP_48Es	FF	LUT
1 pixel	300	0	3	439	280

Performance Estimate

The following table summarizes the performance of RGB to GRAY for different configurations, as generated using the Vivado HLS 2019.1 version tool for the Xczu9eg-ffvb1156-1-i-es1, to process a HD (1080x1920) image.

Table 155: **RGB2GRAY Function Performance Estimate Summary**

Operating Mode	Latency Estimate
	Max Latency (ms)
1 pixel operation (300 MHz)	6.9

BGR to GRAY

The `bgr2gray` function converts a 3-channel BGR image to GRAY format.

$$Y= 0.299*R+0.587*G+0.114*B$$

Where,

- Y = Gray pixel
- R= Red channel

- G= Green channel
- B= Blue channel

API Syntax

```
template<int SRC_T, int DST_T, int ROWS, int COLS, int NPC=1>
void bgr2gray(xf::Mat<SRC_T, ROWS, COLS, NPC> & _src, xf::Mat<DST_T, ROWS,
COLS, NPC> & _dst)
```

Parameter Descriptions

The following table describes the template and the function parameters.

Table 156: **bgr2gray Function Parameter Descriptions**

Parameter	Description
SRC_T	Input pixel type. Only 8-bit, unsigned, 3-channel is supported (XF_8UC3).
DST_T	Output pixel type. Only 8-bit, unsigned,1-channel is supported (XF_8UC1).
ROWS	Maximum height of input and output image. Must be multiple of 8.
COLS	Maximum width of input and output image. Must be multiple of 8.
NPC	Number of pixels to be processed per cycle.
_src	BGR input image
_dst	GRAY output image

Resource Utilization

The following table summarizes the resource utilization of BGR to GRAY for different configurations, as generated in the Vivado HLS 2019.1 version tool for the Xczu9eg-ffvb1156-1-i-es1 FPGA, to process a HD (1080x1920) image.

Table 157: **bgr2gray Function Resource Utilization Summary**

Operating Mode	Operating Frequency (MHz)	Utilization Estimate			
		BRAM_18K	DSP_48Es	FF	LUT
1 pixel	300	0	3	439	280

Performance Estimate

The following table summarizes the performance of BGR to GRAY for different configurations, as generated using the Vivado HLS 2019.1 version tool for the Xczu9eg-ffvb1156-1-i-es1, to process a grayscale HD (1080x1920) image.

Table 158: **bgr2gray Function Performance Estimate Summary**

Operating Mode	Latency Estimate
	Max Latency (ms)
1 pixel operation (300 MHz)	6.9

GRAY to RGB

The `gray2rgb` function converts a gray intensity image to RGB color format.

R<-Y, G<-Y, B<-Y

- Y = Gray pixel
- R= Red channel
- G= Green channel
- B= Blue channel

API Syntax

```
template<int SRC_T,int DST_T,int ROWS,int COLS,int NPC=1>void
gray2rgb(xf::Mat<SRC_T, ROWS, COLS, NPC> & _src,xf::Mat<DST_T, ROWS, COLS,
NPC> & _dst)
```

Parameter Descriptions

The following table describes the template and the function parameters.

Table 159: **gray2rgb Function Parameter Descriptions**

Parameter	Description
SRC_T	Input pixel type. Only 8-bit, unsigned, 1-channel is supported (XF_8UC1).
DST_T	Output pixel type. Only 8-bit, unsigned, 3-channel is supported (XF_8UC3).
ROWS	Maximum height of input and output image. Must be multiple of 8.
COLS	Maximum width of input and output image. Must be multiple of 8.
NPC	Number of pixels to be processed per cycle.
_src	GRAY input image.
_dst	RGB output image.

Resource Utilization

The following table summarizes the resource utilization of gray2rgb for different configurations, as generated in the Vivado HLS 2019.1 version tool for the Xczu9eg-ffvb1156-1-i-es1 FPGA, to process a HD (1080x1920) image.

Table 160: **gray2rgb Function Resource Utilization Summary**

Operating Mode	Operating Frequency (MHz)	Utilization Estimate			
		BRAM_18K	DSP_48Es	FF	LUT
1 pixel	300	0	0	156	184

Performance Estimate

The following table summarizes the performance of gray2rgb for different configurations, as generated using the Vivado HLS 2019.1 version tool for the Xczu9eg-ffvb1156-1-i-es1, to process a grayscale HD (1080x1920) image.

Table 161: **gray2rgb Function Performance Estimate Summary**

Operating Mode	Latency Estimate
	Max Latency (ms)
1 pixel operation (300 MHz)	6.9

GRAY to BGR

The `gray2bgr` function converts a gray intensity image to RGB color format.

R<-Y, G<-Y, B<-Y

Where,

- Y = Gray pixel
- R= Red channel
- G= Green channel
- B= Blue channel

API Syntax

```
template<int SRC_T,int DST_T,int ROWS,int COLS,int NPC=1>
void gray2bgr(xf::Mat<SRC_T, ROWS, COLS, NPC> & _src,xf::Mat<DST_T, ROWS,
COLS, NPC> & _dst)
```

Parameter Descriptions

The following table describes the template and the function parameters.

Table 162: **gray2bgr Function Parameter Descriptions**

Parameter	Description
SRC_T	Input pixel type. Only 8-bit, unsigned, 1-channel is supported (XF_8UC1).
DST_T	Output pixel type. Only 8-bit, unsigned, 3-channel is supported (XF_8UC3).
ROWS	Maximum height of input and output image. Must be multiple of 8.
COLS	Maximum width of input and output image. Must be multiple of 8.
NPC	Number of pixels to be processed per cycle;
_src	GRAY input image.
_dst	BGR output image.

Resource Utilization

The following table summarizes the resource utilization of gray2bgr for different configurations, as generated in the Vivado HLS 2019.1 version tool for the Xczu9eg-ffvb1156-1-i-es1 FPGA, to process a HD (1080x1920) image.

Table 163: **gray2bgr Function Resource Utilization Summary**

Operating Mode	Operating Frequency (MHz)	Utilization Estimate			
		BRAM_18K	DSP_48Es	FF	LUT
1 pixel	300	0	0	156	184

Performance Estimate

The following table summarizes the performance of gray2bgr for different configurations, as generated using the Vivado HLS 2019.1 version tool for the Xczu9eg-ffvb1156-1-i-es1, to process a HD (1080x1920) image.

Table 164: **gray2bgr Function Performance Estimate Summary**

Operating Mode	Latency Estimate
	Max Latency (ms)
1 pixel operation (300 MHz)	6.9

HLS to RGB/BGR

The `hls2(rgb/bgr)` function converts HLS color space to 3-channel RGB/BGR image.

$$C = (1 - |2L - 1|) X\ S_{HSL}$$

$$H' = \frac{H}{60}\circ$$

$$X = C\ X\ (1 - |H' mod2 - 1\ |)$$

$$(R_1, G_1, B_1) = \begin{cases} (0,0,0) & if\ H\ is\ undefined \\ (C, X, 0) & if\ 0 \leq H' \leq 1 \\ (X, C, 0) & if\ 1 \leq H' \leq 2 \\ (0, C, X) & if\ 2 \leq H' \leq 3 \\ (0, X, C) & if\ 3 \leq H' \leq 4 \\ (X, 0, C) & if\ 4 \leq H' \leq 5 \\ (C, 0, X) & if\ 5 \leq H' \leq 6 \end{cases}$$

$$m = L - \frac{C}{2}$$

$$(R, G, B) = (R_1 + m, G_1 + m, B_1 + m)$$

API Syntax

```
template<int SRC_T,int DST_T,int ROWS,int COLS,int NPC=1>void
hls2rgb(xf::Mat<SRC_T, ROWS, COLS, NPC> & _src,xf::Mat<DST_T, ROWS, COLS,
NPC> & _dst)template<int SRC_T,int DST_T,int ROWS,int COLS,int NPC=1>void
hls2bgr(xf::Mat<SRC_T, ROWS, COLS, NPC> & _src,xf::Mat<DST_T, ROWS, COLS,
NPC> & _dst)
```

Parameter Descriptions

The following table describes the template and the function parameters.

Table 165: **HLS2RGB/BGR Function Parameter Descriptions**

Parameter	Description
SRC_T	Input pixel type. Only 8-bit, unsigned, 3-channel is supported (XF_8UC3).
DST_T	Output pixel type. Only 8-bit, unsigned, 3-channel is supported (XF_8UC3).
ROWS	Maximum height of input and output image. Must be multiple of 8.
COLS	Maximum width of input and output image. Must be multiple of 8.
NPC	Number of pixels to be processed per cycle.
_src	HLS input image.
_dst	RGB/BGR output image.

Resource Utilization

The following table summarizes the resource utilization of HLS2RGB/BGRR for different configurations, as generated in the Vivado HLS 2019.1 version tool for the Xczu9eg-ffvb1156-1-i-es1 FPGA, to process a HD (1080x1920) image.

Table 166: **HLS2RGB/BGR Function Resource Utilization Summary**

Operating Mode	Operating Frequency (MHz)	Utilization Estimate			
		BRAM_18K	DSP_48Es	FF	LUT
1 pixel	300	0	3	4366	3096

Performance Estimate

The following table summarizes the performance of HLS2RGB/BGR for different configurations, as generated using the Vivado HLS 2019.1 version tool for the Xczu9eg-ffvb1156-1-i-es1, to process a HD (1080x1920) image.

Table 167: **HLS2RGB/BGR Function Performance Estimate Summary**

Operating Mode	Latency Estimate
	Max Latency (ms)
1 pixel operation (300 MHz)	6.9

RGB to XYZ

The `rgb2xyz` function converts a 3-channel RGB image to XYZ color space.

$$\begin{bmatrix} X \\ Y \\ Z \end{bmatrix} = \begin{bmatrix} 0.412453 & 0.357580 & 0.180423 \\ 0.212671 & 0.715160 & 0.072169 \\ 0.019334 & 0.119193 & 0.950227 \end{bmatrix} \cdot \begin{bmatrix} R \\ G \\ B \end{bmatrix}$$

- R= Red channel
- G= Green channel
- B= Blue channel

API Syntax

```
template<int SRC_T,int DST_T,int ROWS,int COLS,int NPC=1>void
rgb2xyz(xf::Mat<SRC_T, ROWS, COLS, NPC> & _src,xf::Mat<DST_T, ROWS, COLS,
NPC> & _dst)
```

Parameter Descriptions

The following table describes the template and the function parameters.

Table 168: **RGB2XYZ Function Parameter Descriptions**

Parameter	Description
SRC_T	Input pixel type. Only 8-bit, unsigned, 3-channel is supported (XF_8UC3).

Table 168: **RGB2XYZ Function Parameter Descriptions** *(cont'd)*

Parameter	Description
DST_T	Output pixel type. Only 8-bit, unsigned, 3-channel is supported. (XF_8UC3).
ROWS	Maximum height of input and output image. Must be multiple of 8.
COLS	Maximum width of input and output image. Must be multiple of 8.
NPC	Number of pixels to be processed per cycle.
_src	RGB input image.
_dst	XYZ output image.

Resource Utilization

The following table summarizes the resource utilization of RGB to XYZ for different configurations, as generated in the Vivado HLS 2019.1 version tool for the Xczu9eg-ffvb1156-1-i-es1 FPGA, to process a HD (1080x1920) image.

Table 169: **RGB2XYZ Function Resource Utilization Summary**

Operating Mode	Operating Frequency (MHz)	Utilization Estimate			
		BRAM_18K	DSP_48Es	FF	LUT
1 pixel	300	0	8	644	380

Performance Estimate

The following table summarizes the performance of RGB to XYZ for different configurations, as generated using the Vivado HLS 2019.1 version tool for the Xczu9eg-ffvb1156-1-i-es1, to process a HD (1080x1920) image.

Table 170: **RGB2XYZ Function Performance Estimate Summary**

Operating Mode	Latency Estimate
	Max Latency (ms)
1 pixel operation (300 MHz)	6.9

BGR to XYZ

The `bgr2xyz` function converts a 3-channel BGR image to XYZ color space.

$$\begin{bmatrix} X \\ Y \\ Z \end{bmatrix} = \begin{bmatrix} 0.412453 & 0.357580 & 0.180423 \\ 0.212671 & 0.715160 & 0.072169 \\ 0.019334 & 0.119193 & 0.950227 \end{bmatrix} \cdot \begin{bmatrix} B \\ G \\ R \end{bmatrix}$$

- R= Red channel

- G= Green channel
- B= Blue channel

API Syntax

```
template<int SRC_T,int DST_T,int ROWS,int COLS,int NPC=1>void
bgr2xyz(xf::Mat<SRC_T, ROWS, COLS, NPC> & _src,xf::Mat<DST_T, ROWS, COLS,
NPC> & _dst)
```

Parameter Descriptions

The following table describes the template and the function parameters.

Table 171: **RGB2XYZ Function Parameter Descriptions**

Parameter	Description
SRC_T	Input pixel type. Only 8-bit, unsigned, 3-channel is supported (XF_8UC3).
DST_T	Output pixel type. Only 8-bit, unsigned, 3-channel is supported (XF_8UC3).
ROWS	Maximum height of input and output image. Must be a multiple of 8.
COLS	Maximum width of input and output image. Must be a multiple of 8.
NPC	Number of pixels to be processed per cycle.
_src	BGR input image.
_dst	XYZ output image.

Resource Utilization

The following table summarizes the resource utilization of BGR to XYZ for different configurations, as generated in the Vivado HLS 2019.1 version tool for the Xczu9eg-ffvb1156-1-i-es1 FPGA, to process a HD (1080x1920) image.

Table 172: **BGR2XYZ Function Resource Utilization Summary**

Operating Mode	Operating Frequency (MHz)	Utilization Estimate			
		BRAM_18K	DSP_48Es	FF	LUT
1 pixel	300	0	8	644	380

Performance Estimate

The following table summarizes the performance of BGR to XYZ for different configurations, as generated using the Vivado HLS 2019.1 version tool for the Xczu9eg-ffvb1156-1-i-es1, to process a HD (1080x1920) image.

Table 173: **BGR2XYZ Function Performance Estimate Summary**

Operating Mode	Latency Estimate
	Max Latency (ms)
1 pixel operation (300 MHz)	6.9

RGB/BGR to YCrCb

The `(rgb/bgr)2ycrcb` function converts a 3-channel RGB image to YCrCb color space.

- Y = 0.299*R + 0.587*G + 0.114*B
- Cr= (R-Y)*0.713+delta
- Cb= (B-Y)*0.564+delta

$$delta = \begin{cases} 128 & for\ 8\text{ - }bit\ images \\ 32768 & for\ 16\text{ - }bit\ images \\ 0.5 & for\ floating\ point\ images \end{cases}$$

API Syntax

```
template<int SRC_T,int DST_T,int ROWS,int COLS,int NPC=1>void
rgb2ycrcb(xf::Mat<SRC_T, ROWS, COLS, NPC> & _src,xf::Mat<DST_T, ROWS, COLS,
NPC> & _dst)
```

```
template<int SRC_T,int DST_T,int ROWS,int COLS,int NPC=1>void
bgr2ycrcb(xf::Mat<SRC_T, ROWS, COLS, NPC> & _src,xf::Mat<DST_T, ROWS, COLS,
NPC> & _dst)
```

Parameter Descriptions

The following table describes the template and the function parameters.

Table 174: **RGB/BGR2YCrCb Function Parameter Descriptions**

Parameter	Description
SRC_T	Input pixel type. Only 8-bit, unsigned, 3-channel is supported (XF_8UC3)
DST_T	Output pixel type. Only 8-bit, unsigned, 3-channel is supported (XF_8UC3)
ROWS	Maximum height of input and output image. Must be multiple of 8.
COLS	Maximum width of input and output image. Must be multiple of 8.
NPC	Number of pixels to be processed per cycle
_src	RGB/BGR input image
_dst	YCrCb output image

Resource Utilization

The following table summarizes the resource utilization of RGB/BGR2YCrCb for different configurations, as generated in the Vivado HLS 2019.1 version tool for the Xczu9eg-ffvb1156-1-i-es1 FPGA, to process a HD (1080x1920) image.

Table 175: **RGB/BGR2YCrCb Function Resource Utilization Summary**

Operating Mode	Operating Frequency (MHz)	Utilization Estimate			
		BRAM_18K	DSP_48Es	FF	LUT
1 pixel	300	0	5	660	500

Performance Estimate

The following table summarizes the performance of RGB/BGR2YCrCb for different configurations, as generated using the Vivado HLS 2019.1 version tool for the Xczu9eg-ffvb1156-1-i-es1, to process a HD (1080x1920) image.

Table 176: **RGB/BGR2YCrCb Function Performance Estimate Summary**

Operating Mode	Latency Estimate
	Max Latency (ms)
1 pixel operation (300 MHz)	6.9

RGB/BGR to HSV

The `(rgb/bgr)2hsv` function converts a 3-channel RGB image to HSV color space.

$$V = \max\ (R,\ G,\ B)$$

$$S = \begin{cases} \dfrac{\text{V - min}(R,\ G,\ B)}{V} & if\ V \neq 0 \\ 0 & otherwise \end{cases}$$

$$H = \begin{cases} 60(G - B)/\ \ (V - min(R,\ G,\ B)) & if\ \ V = R \\ 120 + 60(B - R)/\ \ (V - min(R,\ G,\ B)) & if\ V = G \\ 240 + 60(R - G)/\ \ (V - min(R,\ G,\ B)) & if\ V = B \end{cases}$$

$$delta = \begin{cases} 128 & for\ 8 - bit\ images \\ 32768 & for\ 16 - bit\ images \\ 0.5 & for\ floating\ point\ images \end{cases}$$

API Syntax

```
template<int SRC_T,int DST_T,int ROWS,int COLS,int NPC=1>void
rgb2hsv(xf::Mat<SRC_T, ROWS, COLS, NPC> & _src,xf::Mat<DST_T, ROWS, COLS,
NPC> & _dst)
```

```
template<int SRC_T,int DST_T,int ROWS,int COLS,int NPC=1> void
bgr2hsv(xf::Mat<SRC_T, ROWS, COLS, NPC> & _src,xf::Mat<DST_T, ROWS, COLS,
NPC> & _dst)
```

Parameter Descriptions

The following table describes the template and the function parameters.

Table 177: **RGB/BGR2HSV Function Parameter Descriptions**

Parameter	Description
SRC_T	Input pixel type. Only 8-bit, unsigned, 3-channel is supported (XF_8UC3).
DST_T	Output pixel type. Only 8-bit, unsigned, 3-channel is supported (XF_8UC3).
ROWS	Maximum height of input and output image. Must be multiple of 8.
COLS	Maximum width of input and output image. Must be multiple of 8.
NPC	Number of pixels to be processed per cycle
_src	RGB/BGR input image
_dst	HSV output image

Resource Utilization

The following table summarizes the resource utilization of RGB/BGR2HSV for different configurations, as generated in the Vivado HLS 2019.1 version tool for the Xczu9eg-ffvb1156-1-i-es1 FPGA, to process a HD (1080x1920) image.

Table 178: **RGB/BGR2HSV Function Resource Utilization Summary**

Operating Mode	Operating Frequency (MHz)	Utilization Estimate			
		BRAM_18K	DSP_48Es	FF	LUT
1 pixel	300	6	8	1582	1274

Performance Estimate

The following table summarizes the performance of RGB/BGR2HSV for different configurations, as generated using the Vivado HLS 2019.1 version tool for the Xczu9eg-ffvb1156-1-i-es1, to process a HD (1080x1920) image.

Table 179: **RGB/BGR2HSV Function Performance Estimate Summary**

Operating Mode	Latency Estimate
	Max Latency (ms)
1 pixel operation (300 MHz)	6.9

RGB/BGR to HLS

The `(rgb/bgr)2hls` function converts a 3-channel RGB image to HLS color space.

$$Vmax = \max(R,\ G,\ B)$$

$$Vmin = \min(R,\ G,\ B)$$

$$L = \frac{Vmax + Vmin}{2}$$

$$S = \begin{cases} \dfrac{Vmax - Vmin}{Vmax + Vmin} & if\ l < 0.5 \\ \dfrac{Vmax - Vmin}{2 - (Vmax + Vmin?)} & if\ L \geq 0.5 \end{cases}$$

$$H = \begin{cases} \dfrac{60(G - B)}{S} & if\ \ Vmax = R \\ 120 + \dfrac{60(B - R)}{S} & if\ Vmax = G \\ 240 + \dfrac{60(R - G)}{S} & if\ Vmax = B \end{cases}$$

API Syntax

```
template<int SRC_T,int DST_T,int ROWS,int COLS,int NPC=1>void
rgb2hls(xf::Mat<SRC_T, ROWS, COLS, NPC> & _src,xf::Mat<DST_T, ROWS, COLS,
NPC> & _dst)template<int SRC_T,int DST_T,int ROWS,int COLS,int NPC=1>void
bgr2hls(xf::Mat<SRC_T, ROWS, COLS, NPC> & _src,xf::Mat<DST_T, ROWS, COLS,
NPC> & _dst)
```

Parameter Descriptions

The following table describes the template and the function parameters.

Table 180: **RGB/BGR2HLS Function Parameter Descriptions**

Parameter	Description
SRC_T	Input pixel type. Only 8-bit, unsigned, 3-channel is supported (XF_8UC3).
DST_T	Output pixel type. Only 8-bit, unsigned, 3-channel is supported (XF_8UC3).
ROWS	Maximum height of input and output image. Must be multiple of 8.
COLS	Maximum width of input and output image. Must be multiple of 8.
NPC	Number of pixels to be processed per cycle.
_src	RGB/BGR input image.

Table 180: **RGB/BGR2HLS Function Parameter Descriptions** *(cont'd)*

Parameter	Description
_dst	HLS output image.

Resource Utilization

The following table summarizes the resource utilization of RGB/BGR2HLS for different configurations, as generated in the Vivado HLS 2019.1 version tool for the Xczu9eg-ffvb1156-1-i-es1 FPGA, to process a HD (1080x1920) image.

Table 181: **RGB/BGR2HLS Function Resource Utilization Summary**

Operating Mode	Operating Frequency (MHz)	Utilization Estimate			
		BRAM_18K	DSP_48Es	FF	LUT
1 pixel	300	0	3	4366	3096

Performance Estimate

The following table summarizes the performance of RGB/BGR2HLS for different configurations, as generated using the Vivado HLS 2019.1 version tool for the Xczu9eg-ffvb1156-1-i-es1, to process a HD (1080x1920) image.

Table 182: **RGB/BGR2HLS Function Performance Estimate Summary**

Operating Mode	Latency Estimate
	Max Latency (ms)
1 pixel operation (300 MHz)	6.9

YCrCb to RGB/BGR

The `ycrcb2(rgb/bgr)` function converts YCrCb color space to 3-channel RGB/BGR image.

Where,

- R= Y+1.403*(Cr-delta)
- G= Y-0.714*(Cr-delta)-0.344*(cb-delta)
- B= Y+1.773+(Cb-delta)

$$delta = \begin{cases} 128 & for\ 8\text{-}bit\ images \\ 32768 & for\ 16\text{-}bit\ images \\ 0.5 & for\ floating\ point\ images \end{cases}$$

API Syntax

```
template<int SRC_T,int DST_T,int ROWS,int COLS,int NPC=1>void
ycrcb2rgb(xf::Mat<SRC_T, ROWS, COLS, NPC> & _src,xf::Mat<DST_T, ROWS, COLS,
NPC> & _dst)
```

```
template<int SRC_T,int DST_T,int ROWS,int COLS,int NPC=1>void
ycrcb2bgr(xf::Mat<SRC_T, ROWS, COLS, NPC> & _src,xf::Mat<DST_T, ROWS, COLS,
NPC> & _dst)
```

Parameter Descriptions

The following table describes the template and the function parameters.

Table 183: **YCrCb2RGB/BGR Function Parameter Descriptions**

Parameter	Description
SRC_T	Input pixel type. Only 8-bit, unsigned, 3-channel is supported (XF_8UC3).
DST_T	Output pixel type. Only 8-bit, unsigned, 3-channel is supported (XF_8UC3).
ROWS	Maximum height of input and output image. Must be a multiple of 8.
COLS	Maximum width of input and output image. Must be a multiple of 8.
NPC	Number of pixels to be processed per cycle.
_src	YCrCb input image.
_dst	RGB/BGR output image.

Resource Utilization

The following table summarizes the resource utilization of YCrCb2RGB/BGR for different configurations, as generated in the Vivado HLS 2019.1 version tool for the Xczu9eg-ffvb1156-1-i-es1 FPGA, to process a HD (1080x1920) image.

Table 184: **YCrCb2RGB/BGR Function Resource Utilization Summary**

Operating Mode	Operating Frequency (MHz)	Utilization Estimate			
		BRAM_18K	DSP_48Es	FF	LUT
1 pixel	300	0	4	538	575

Performance Estimate

The following table summarizes the performance of YCrCb2RGB/BGR for different configurations, as generated using the Vivado HLS 2019.1 version tool for the Xczu9eg-ffvb1156-1-i-es1, to process a HD (1080x1920) image.

Table 185: **YCrCb2RGB/BGR Function Performance Estimate Summary**

Operating Mode	Latency Estimate
	Max Latency (ms)
1 pixel operation (300 MHz)	6.9

HSV to RGB/BGR

The `hsv2(rgb/bgr)` function converts HSV color space to 3-channel RGB/BGR image.

$$C = V\ X\ S_{HSV}$$

$$H' = \frac{H}{60}\circ$$

$$X = C\ X\ (1 - |H' mod2 - 1\ |)$$

$$(R_1,\ G_1,\ B_1) = \begin{cases} (0,0,\ 0) & if\ H\ is\ undefined \\ (C,\ X,\ 0) & if\ 0 \le H' \le 1 \\ (X,\ C,\ 0) & if\ 1 \le H' \le 2 \\ (0,\ C,\ X) & if\ 2 \le H' \le 3 \\ (0,\ X,\ C) & if\ 3 \le H' \le 4 \\ (X,\ 0,\ C) & if\ 4 \le H' \le 5 \\ (C,\ 0,\ X) & if\ 5 \le H' \le 6 \end{cases}$$

$$m = V - C$$

$$(R,\ G,\ B) = (R_1 + m,\ G_1 + m\ ,\ B_1 + m)$$

API Syntax

```
template<int SRC_T,int DST_T,int ROWS,int COLS,int NPC=1>void
hsv2rgb(xf::Mat<SRC_T, ROWS, COLS, NPC> & _src,xf::Mat<DST_T, ROWS, COLS,
NPC> & _dst)
```

```
template<int SRC_T,int DST_T,int ROWS,int COLS,int NPC=1>void
hsv2bgr(xf::Mat<SRC_T, ROWS, COLS, NPC> & _src,xf::Mat<DST_T, ROWS, COLS,
NPC> & _dst)
```

Parameter Descriptions

The following table describes the template and the function parameters.

Table 186: **HSV2RGB/BGR Function Parameter Descriptions**

Parameter	Description
SRC_T	Input pixel type. Only 8-bit, unsigned, 3-channel is supported (XF_8UC3)
DST_T	Output pixel type. Only 8-bit, unsigned, 3-channel is supported (XF_8UC3)
ROWS	Maximum height of input and output image. Must be multiple of 8.

Table 186: **HSV2RGB/BGR Function Parameter Descriptions** *(cont'd)*

Parameter	Description
COLS	Maximum width of input and output image. Must be multiple of 8.
NPC	Number of pixels to be processed per cycle
_src	HSV input image
_dst	RGB/BGR output image

Resource Utilization

The following table summarizes the resource utilization of HSV2RGB/BGRR for different configurations, as generated in the Vivado HLS 2019.1 version tool for the Xczu9eg-ffvb1156-1-i-es1 FPGA, to process a HD (1080x1920) image.

Table 187: **HSV2RGB/BGR Function Resource Utilization Summary**

Operating Mode	Operating Frequency (MHz)	Utilization Estimate			
		BRAM_18K	DSP_48Es	FF	LUT
1 pixel	300	0	8	1543	1006

Performance Estimate

The following table summarizes the performance of HSV2RGB/BGR for different configurations, as generated using the Vivado HLS 2019.1 version tool for the Xczu9eg-ffvb1156-1-i-es1, to process a HD (1080x1920) image.

Table 188: **HSV2RGB/BGR Function Performance Estimate Summary**

Operating Mode	Latency Estimate
	Max Latency (ms)
1 pixel operation (300 MHz)	6.9

NV12/NV21 to RGB/ BGR

The `nv122rgb/nv122bgr/nv212rgb/nv212bgr` converts NV12 image format to a 3-channel RGB/BGR image. The inputs to the function are separate Y and UV planes. NV12 holds sub sampled data, Y plane is sampled at unit rate, and 1 U and 1 V value each for every 2x2 Y values. To generate the RGB data, each U and V value is duplicated (2x2) times.

API Syntax

NV122RGB:

```
template<int SRC_T,int UV_T,int DST_T,int ROWS,int COLS,int NPC=1,int
NPC_UV=1>void nv122rgb(xf::Mat<SRC_T, ROWS, COLS, NPC> &
src_y,xf::Mat<UV_T, ROWS/2, COLS/2, NPC_UV> & src_uv,xf::Mat<DST_T, ROWS,
COLS, NPC> & _dst0)
```

NV122BGR:

```
template<int SRC_T,int UV_T,int DST_T,int ROWS,int COLS,int NPC=1,int
NPC_UV=1>void nv122bgr(xf::Mat<SRC_T, ROWS, COLS, NPC> &
src_y,xf::Mat<UV_T, ROWS/2, COLS/2, NPC_UV> & src_uv,xf::Mat<DST_T, ROWS,
COLS, NPC> & _dst0)
```

NV212RGB:

```
template<int SRC_T,int UV_T,int DST_T,int ROWS,int COLS,int NPC=1,int
NPC_UV=1>void nv212rgb(xf::Mat<SRC_T, ROWS, COLS, NPC> &
src_y,xf::Mat<UV_T, ROWS/2, COLS/2, NPC_UV> & src_uv,xf::Mat<DST_T, ROWS,
COLS, NPC> & _dst0)
```

NV212BGR:

```
template<int SRC_T,int UV_T,int DST_T,int ROWS,int COLS,int NPC=1,int
NPC_UV=1>void nv212bgr(xf::Mat<SRC_T, ROWS, COLS, NPC> & src_y,
xf::Mat<UV_T, ROWS/2, COLS/2, NPC_UV> & src_uv, xf::Mat<DST_T, ROWS, COLS,
NPC> & _dst0)
```

Parameter Descriptions

The following table describes the template and the function parameters.

Table 189: **Function Parameter Descriptions**

Parameter	Description
SRC_T	Input pixel type. Only 8-bit,unsigned, 1-channel is supported (XF_8UC1).
UV_T	Input pixel type. Only 8-bit, unsigned, 2-channel is supported (XF_8UC2).
ROWS	Maximum height of input and output image
COLS	Maximum width of input and output image. Must be a multiple of NPC for N pixel mode.
NPC	Number of Y Pixels to be processed per cycle. Possible options are XF_NPPC1,XF_NPPC2,XF_NPPC4 and XF_NPPC8.
NPC_UV	Number of UV Pixels to be processed per cycle. Possible options are XF_NPPC1,XF_NPPC2 and XF_NPPC4.
src_y	Y input image of size(ROWS, COLS)
src_uv	UV output image of size (ROWS/2, COLS/2).
_dst0	Output UV image of size (ROWS, COLS).

Resource Utilization

The following table summarizes the resource utilization of NV12/NV21 to RGB/ BGR function in Normal mode (1 pixel), as generated in the Vivado HLS 2019.1 tool for the Xilinx xczu9eg-ffvb1156-2-i-es2 FPGA to process a HD (1080x1920) image.

Operating Mode	Operating Frequency (MHz)	Utilization Estimate				
		BRAM_18K	DSP_48Es	FF	LUT	CLB
1 pixel	300	2	5	339	289	76

Performance Estimate

The following table summarizes the performance of the kernel in single pixel configuration as generated using Vivado HLS 2018.3 tool for the Xilinx xczu9eg-ffvb1156-2-i-es2 FPGA to process a HD (1080x1920) image.

Table 190: **Performance Estimate Summary**

Operating Mode	Latency Estimate
	Max Latency (ms)
1 pixel operation (300 MHz)	6.9

NV12 to NV21/NV21 to NV12

The nv122nv21/nv212nv12 function converts a NV12 (YUV4:2:0) to NV21 (YUV4:2:0) or vice versa, where 8-bit Y plane followed by an interleaved U/V plane with 2x2 sub-sampling.

API Syntax

NV122NV21:

```
template<int SRC_Y,int SRC_UV,int ROWS,int COLS,int NPC=1,int NPC_UV=1>
void nv122nv21(xf::Mat<SRC_Y, ROWS, COLS, NPC> & _y,xf::Mat<SRC_UV, ROWS/2,
COLS/2, NPC_UV> & _uv,xf::Mat<SRC_Y, ROWS, COLS, NPC> &
out_y,xf::Mat<SRC_UV, ROWS/2, COLS/2, NPC_UV> & out_uv)
```

NV212NV12:

```
template<int SRC_Y, int SRC_UV, int ROWS, int COLS, int NPC=1,int
NPC_UV=1>void nv212nv12(xf::Mat<SRC_Y, ROWS, COLS, NPC> & _y,
xf::Mat<SRC_UV, ROWS/2, COLS/2, NPC_UV> & _uv, xf::Mat<SRC_Y, ROWS, COLS,
NPC> & out_y, xf::Mat<SRC_UV, ROWS/2, COLS/2, NPC_UV> & out_uv)
```

Parameter Descriptions

The following table describes the template and the function parameters.

Table 191: **Function Parameter Descriptions**

Parameter	Description
SRC_Y	Input Y pixel type. Only 8-bit, unsigned, 1-channel is supported (XF_8UC1)
SRC_UV	Input UV pixel type. Only 8-bit, unsigned, 2-channel is supported (XF_8UC2)
ROWS	Maximum height of input and output image
COLS	Maximum width of input and output image. Must be multiple of N.
NPC_Y	Number of Y pixels to be processed per cycle. Possible options are XF_NPPC1,XF_NPPC2,XF_NPPC4 and XF_NPPC8.
NPC_UV	Number of UV Pixels to be processed per cycle. Possible options are XF_NPPC1,XF_NPPC2 and XF_NPPC4.
_y	Y input image
_uv	UV input image
out_y	Y output image
out_uv	UV output image

Resource Utilization

The following table summarizes the resource utilization of NV122NV21/NV212NV12 function in Normal mode (1-Pixel), as generated in the Vivado HLS 2019.1 tool for the Xilinx xczu9eg-ffvb1156-2-i-es2 FPGA to process a HD (1080x1920) image.

Operating Mode	Operating Frequency (MHz)	Utilization Estimate				
		BRAM_18K	DSP_48Es	FF	LUT	CLB
1 pixel	300	0	0	258	161	61

Performance Estimate

The following table summarizes the performance of the kernel in single pixel configuration as generated using Vivado HLS 2019.1 tool for the Xilinx xczu9eg-ffvb1156-2-i-es2 FPGA to process a HD (1080x1920) image.

Table 192: **Performance Estimate Summary**

Operating Mode	Latency Estimate
	Max Latency (ms)
1 pixel operation (300 MHz)	6.9

NV12/NV21 to UYVY/YUYV

The `NV12/NV21 to UYVY/YUYV` function converts a NV12/NV21 (YUV4:2:0) image to a single-channel YUYV/UYVY (YUV 4:2:2) image format. YUYV is a sub-sampled format. YUYV/UYVY is represented in 16-bit values whereas, RGB is represented in 24-bit values.

API Syntax

NV122UYVY:

```
template<int SRC_Y, int SRC_UV, int DST_T, int ROWS, int COLS, int
NPC=1,int NPC_UV=1>void nv122uyvy(xf::Mat<SRC_Y, ROWS, COLS, NPC> &
_y,xf::Mat<SRC_UV, ROWS/2, COLS/2, NPC_UV> & _uv,xf::Mat<DST_T, ROWS, COLS,
NPC> & _dst)
```

NV122YUYV:

```
template<int SRC_Y, int SRC_UV, int DST_T, int ROWS, int COLS, int
NPC=1,int NPC_UV=1>void nv122yuyv(xf::Mat<SRC_Y, ROWS, COLS, NPC> & _y,
xf::Mat<SRC_UV, ROWS/2, COLS/2, NPC_UV> & _uv, xf::Mat<DST_T, ROWS, COLS,
NPC> & _dst)
```

NV212UYVY:

```
template<int SRC_Y, int SRC_UV, int DST_T, int ROWS, int COLS, int
NPC=1,int NPC_UV=1>void nv212uyvy(xf::Mat<SRC_Y, ROWS, COLS, NPC> & _y,
xf::Mat<SRC_UV, ROWS/2, COLS/2, NPC_UV> & _uv,xf::Mat<DST_T, ROWS, COLS,
NPC> & _dst)
```

NV212YUYV:

```
 template<int SRC_Y, int SRC_UV, int DST_T,int ROWS, int COLS, int
NPC=1,int NPC_UV=1>void nv212yuyv(xf::Mat<SRC_Y, ROWS, COLS, NPC> & _y,
xf::Mat<SRC_UV, ROWS/2, COLS/2, NPC_UV> & _uv, xf::Mat<DST_T, ROWS, COLS,
NPC> & _dst)
```

Parameter Descriptions

The following table describes the template and the function parameters.

Table 193: **Function Parameter Descriptions**

Parameter	Description
SRC_Y	Input Y image pixel type. Only 8-bit, unsigned, 1-channel is supported (XF_8UC1).
SRC_UV	Input UV image pixel type. Only 8-bit, unsigned, 2-channel is supported (XF_8UC2).
DST_T	Output pixel type. Only 16-bit, unsigned, 1-channel is supported (XF_16UC1).
ROWS	Maximum height of input and output image.
COLS	Maximum width of input and output image. Must be multiple of NPC.
NPC	Number of pixels to be processed per cycle. Possible options are XF_NPPC1,XF_NPPC2,XF_NPPC4 and XF_NPPC8.
NPC_UV	Number of pixels to be processed per cycle. Possible options are XF_NPPC1,XF_NPPC2 and XF_NPPC4.
_y	Y input image
_uv	UV input image
_dst	UYVY/YUYV output image

Resource Utilization

The following table summarizes the resource utilization of `NV12/NV21 to UYVY/YUYV` function in Normal mode(1-Pixel), as generated in the Vivado HLS 2019.1 tool for the Xilinx xczu9eg-ffvb1156-2-i-es2 FPGA to process a HD (1080x1920) image.

Operating Mode	Operating Frequency (MHz)	Utilization Estimate				
		BRAM_18K	DSP_48Es	FF	LUT	CLB
1 pixel	300	1	0	337	201	64

Performance Estimate

The following table summarizes the performance of the kernel in single pixel configuration as generated using Vivado HLS 2019.1 tool for the Xilinx xczu9eg-ffvb1156-2-i-es2 FPGA to process a HD (1080x1920) image.

Table 194: **Performance Estimate Summary**

Operating Mode	Latency Estimate
	Max Latency (ms)
1 pixel operation (300 MHz)	6.9

UYVY/YUYV to RGB/BGR

The yuyv2rgb/yuyv2bgr/uyvy2rgb/uyvy2bgr function converts a single-channel YUYV/UYVY (YUV 4:2:2) image format to a 3- channel RGB/BGR image. YUYV/UYVY is a sub-sampled format, a set of YUYV/UYVY values gives 2 RGB pixel values. YUYV/UYVY is represented in 16-bit values whereas, RGB/BGR is represented in 24-bit values

API Syntax

YUYV2RGB:

```
template<int SRC_T,int DST_T,int ROWS,int COLS,int NPC=1>void
yuyv2rgb(xf::Mat<SRC_T, ROWS, COLS, NPC> & _src,xf::Mat<DST_T, ROWS, COLS,
NPC> & _dst)
```

YUYV2BGR:

```
template<int SRC_T,int DST_T,int ROWS,int COLS,int NPC=1>void
yuyv2bgr(xf::Mat<SRC_T, ROWS, COLS, NPC> & _src,xf::Mat<DST_T, ROWS, COLS,
NPC> & _dst)
```

UYVY2RGB

```
template<int SRC_T,int DST_T,int ROWS,int COLS,int NPC=1>void
uyvy2rgb(xf::Mat<SRC_T, ROWS, COLS, NPC> & _src,xf::Mat<DST_T, ROWS, COLS,
NPC> & _dst)
```

UYVY2BGR:

```
template<int SRC_T,int DST_T,int ROWS,int COLS,int NPC=1>void
uyvy2bgr(xf::Mat<SRC_T, ROWS, COLS, NPC> & _src,xf::Mat<DST_T, ROWS, COLS,
NPC> & _dst)
```

Parameter Descriptions

The following table describes the template and the function parameters.

Table 195: **Function Parameter Descriptions**

Parameter	Description
SRC_T	Input pixel type. Only 16-bit, unsigned,1-channel is supported (XF_16UC1).
DST_T	Output pixel type. Only 8-bit, unsigned, 3-channel is supported (XF_8UC3).
ROWS	Maximum height of input and output image
COLS	Maximum width of input and output image. Must be a multiple of NPC for N pixel mode.
NPC	Number of Y pixels to be processed per cycle. Possible options are XF_NPPC1,XF_NPPC2,XF_NPPC4 and XF_NPPC8.
_src	Input image of size(ROWS, COLS)
_dst	Output image of size (ROWS, COLS).

Resource Utilization

The following table summarizes the resource utilization of `UYVY/YUYV` to `RGB/BGR` function in Normal mode(1-Pixel), as generated in the Vivado HLS 2019.1 tool for the Xilinx xczu9eg-ffvb1156-2-i-es2 FPGA to process a HD (1080x1920) image.

Operating Mode	Operating Frequency (MHz)	Utilization Estimate				
		BRAM_18K	DSP_48Es	FF	LUT	CLB
1 pixel	300	0	6	444	486	109

Performance Estimate

The following table summarizes the performance of the kernel in single pixel configuration as generated using Vivado HLS 2019.1 tool for the Xilinx xczu9eg-ffvb1156-2-i-es2 FPGA to process a HD (1080x1920) image.

Table 196: **Performance Estimate Summary**

Operating Mode	Latency Estimate
	Max Latency (ms)
1 pixel operation (300 MHz)	6.9

UYVY to YUYV/ YUYV to UYVY

The `yuyv2uyvy/uyvy2yuyv` function converts a YUYV (YUV4:2:2) to UYVY (YUV4:2:2) or vice versa, where 8-bit Y plane followed by an interleaved U/V plane with 2x2 sub sampling.

API Syntax

UYVY2YUYV :

```
template<int SRC_T,int DST_T,int ROWS,int COLS,int NPC=1>void
uyvy2yuyv(xf::Mat<SRC_T, ROWS, COLS, NPC> & uyvy,xf::Mat<DST_T, ROWS, COLS,
NPC> & yuyv)
```

YUYV2UYVY:

```
template<int SRC_T,int DST_T,int ROWS,int COLS,int NPC=1>void
yuyv2uyvy(xf::Mat<SRC_T, ROWS, COLS, NPC> & yuyv,xf::Mat<DST_T, ROWS, COLS,
NPC> & uyvy)
```

Parameter Descriptions

The following table describes the template and the function parameters.

Table 197: **Function Parameter Descriptions**

Parameter	Description
SRC_T	Input Y pixel type. Only 16-bit, unsigned, 1-channel is supported (XF_16UC1).
ROWS	Maximum height of input and output image
COLS	Maximum width of input and output image. Must be a multiple of N.
NPC	Number of pixels to be processed per cycle. Possible options are XF_NPPC1,XF_NPPC2,XF_NPPC4 and XF_NPPC8.
yuyv	Input image
uyvy	Output image

Resource Utilization

The following table summarizes the resource utilization of `UYVY` to `YUYV/ YUYV` to `UYVY` function in Normal mode (1 pixel), as generated in the Vivado HLS 2019.1 tool for the Xilinx xczu9eg-ffvb1156-2-i-es2 FPGA.

Operating Mode	Operating Frequency (MHz)	Utilization Estimate				
		BRAM_18K	DSP_48Es	FF	LUT	CLB
1 pixel	300	0	1	368	176	109

Performance Estimate

The following table summarizes the performance of the kernel in single pixel configuration as generated using Vivado HLS 2019.1 tool for the Xilinx xczu9eg-ffvb1156-2-i-es2 FPGA to process a grayscale HD (1080x1920) image.

Table 198: **Performance Estimate Summary**

Operating Mode	Latency Estimate
	Max Latency (ms)
1 pixel operation (300 MHz)	6.9

UYVY/YUYV to NV21

The `UYVY/YUYV2NV21` function converts a single-channel YUYV/UYVY (YUV 4:2:2) image format to NV21 (YUV 4:2:0) format. YUYV/UYVY is a sub-sampled format, 1 set of YUYV/UYVY value gives 2 Y values and 1 U and V value each.

API Syntax

UYVY2NV21:

```
template<int SRC_T,int Y_T,int UV_T,int ROWS,int COLS,int NPC=1,int
NPC_UV=1>void uyvy2nv21(xf::Mat<SRC_T, ROWS, COLS, NPC> & _src,xf::Mat<Y_T,
ROWS, COLS, NPC> & _y_image,xf::Mat<UV_T, ROWS/2, COLS/2, NPC_UV> &
_uv_image)
```

YUYV2NV21:

```
template<int SRC_T,int Y_T,int UV_T,int ROWS,int COLS,int NPC=1,int
NPC_UV=1>void yuyv2nv21(xf::Mat<SRC_T, ROWS, COLS, NPC> & _src,xf::Mat<Y_T,
ROWS, COLS, NPC> & _y_image,xf::Mat<UV_T, ROWS/2, COLS/2, NPC_UV> &
_uv_image)
```

Parameter Descriptions

The following table describes the template and the function parameters.

Table 199: **Function Parameter Descriptions**

Parameter	Description
SRC_T	Input pixel type. Only 16-bit, unsigned,1-channel is supported (XF_16UC1).
Y_T	Output Y image pixel type. Only 8-bit, unsigned, 1-channel is supported (XF_8UC1).

Table 199: **Function Parameter Descriptions** *(cont'd)*

Parameter	Description
UV_T	Output UV image pixel type. Only 8-bit, unsigned, 2-channel is supported (XF_8UC2).
ROWS	Maximum height of input and output image.
COLS	Maximum width of input and output image. Must be multiple of NPC.
NPC	Number of pixels to be processed per cycle; Possible options are XF_NPPC1,XF_NPPC2,XF_NPPC4 and XF_NPPC8.
NPC_UV	Number of U, V Pixels to be processed per cycle; Possible options are XF_NPPC1,XF_NPPC2 and XF_NPPC4.
_src	Input image
_y_image	Y Output image
_uv_image	UV Output image

Resource Utilization

The following table summarizes the resource utilization of `UYVY/YUYV` to `NV21` function in Normal mode (1 pixel), as generated in the Vivado HLS 2019.1 tool for the Xilinx xczu9eg-ffvb1156-2-i-es2 FPGA to process a HD (1080x1920) image.

Operating Mode	Operating Frequency (MHz)	Utilization Estimate				
		BRAM_18K	DSP_48Es	FF	LUT	CLB
1 pixel	300	0	0	215	73	42

Performance Estimate

The following table summarizes the performance of the kernel in single pixel configuration as generated using Vivado HLS 2019.1 tool for the Xilinx xczu9eg-ffvb1156-2-i-es2 FPGA to process a HD (1080x1920) image.

Table 200: **Performance Estimate Summary**

Operating Mode	Latency Estimate
	Max Latency (ms)
1 pixel operation (300 MHz)	6.9

RGB/ BGR to NV12/NV21

The `rgb2nv12/bgr2nv12/rgb2nv21/bgr2nv21` converts a 3-channel RGB/BGR image to NV12/NV21 (4:2:0) format. The function outputs Y plane and interleaved UV/VU plane separately. NV12/NV21 holds the subsampled data, Y is sampled for every RGB/BGR pixel and U, V are sampled once for 2 rows and 2columns (2x2) pixels. UV/VU plane is of (rows/2)*(columns/2) size as U and V values are interleaved.

API Syntax

RGB2NV12

```
template <int SRC_T, int Y_T, int UV_T, int ROWS, int COLS, int NPC=1,int
NPC_UV=1>void rgb2nv12(xf::Mat<SRC_T, ROWS, COLS, NPC> & _src, xf::Mat<Y_T,
ROWS, COLS, NPC> & _y, xf::Mat<UV_T, ROWS/2, COLS/2, NPC_UV> & _uv)
```

BGR2NV12

```
template <int SRC_T, int Y_T, int UV_T, int ROWS, int COLS, int NPC=1,int
NPC_UV=1>void bgr2nv12(xf::Mat<SRC_T, ROWS, COLS, NPC> & _src, xf::Mat<Y_T,
ROWS, COLS, NPC> & _y, xf::Mat<UV_T, ROWS/2, COLS/2, NPC_UV> & _uv)
```

RGB2NV21

```
template <int SRC_T, int Y_T, int UV_T, int ROWS, int COLS, int NPC=1,int
NPC_UV=1>void rgb2nv21(xf::Mat<SRC_T, ROWS, COLS, NPC> & _src, xf::Mat<Y_T,
ROWS, COLS, NPC> & _y, xf::Mat<UV_T, ROWS/2, COLS/2, NPC_UV> & _uv)
```

BGR2NV21

```
template <int SRC_T, int Y_T, int UV_T, int ROWS, int COLS, int NPC=1,int
NPC_UV=1>void bgr2nv21(xf::Mat<SRC_T, ROWS, COLS, NPC> & _src, xf::Mat<Y_T,
ROWS, COLS, NPC> & _y, xf::Mat<UV_T, ROWS/2, COLS/2, NPC_UV> & _uv)
```

Parameter Descriptions

The following table describes the template and the function parameters.

Table 201: **Function Parameter Descriptions**

Parameter	Description
SRC_T	Input pixel type. Only 8-bit, unsigned, 3-channel is supported (XF_8UC3).
Y_T	Output pixel type. Only 8-bit, unsigned, 1-channel is supported (XF_8UC1).
UV_T	Output pixel type. Only 8-bit, unsigned, 2-channel is supported (XF_8UC2).
ROWS	Maximum height of input and output image
COLS	Maximum width of input and output image. Must be a multiple of NPC for N pixel mode.
NPC	Number of Pixels to be processed per cycle. Possible options are XF_NPPC1,XF_NPPC2,XF_NPPC4 and XF_NPPC8.
NPC_UV	Number of Pixels to be processed per cycle. Possible options are XF_NPPC1,XF_NPPC2 and XF_NPPC4
_src	RGB input image of size(ROWS,COLS)
_y	Output Y image of size (ROWS, COLS).
_uv	Output UV image of size (ROWS/2, COLS/2).

Resource Utilization

The following table summarizes the resource utilization of RGB/BGR to NV12/NV21 function in Normal mode (1-Pixel), as generated in the Vivado HLS 2019.1 tool for the Xilinx xczu9eg-ffvb1156-2-i-es2 FPGA to process a HD (1080x1920) image.

Operating Mode	Operating Frequency (MHz)	Utilization Estimate				
		BRAM_18K	DSP_48Es	FF	LUT	CLB
1 pixel	300	0	9	413	279	66

Performance Estimate

The following table summarizes the performance of the kernel in single pixel configuration as generated using Vivado HLS 2019.1 tool for the Xilinx xczu9eg-ffvb1156-2-i-es2 FPGA to process a HD (1080x1920) image.

Table 202: **Performance Estimate Summary**

Operating Mode	Latency Estimate
	Max Latency (ms)
1 pixel operation (300 MHz)	6.9

BGR to RGB / RGB to BGR

The bgr2rgb/rgb2bgr function converts a 3-channel BGR to RGB format or RGB to BGR format.

API Syntax

```
template<int SRC_T,int DST_T,int ROWS,int COLS,int NPC=1>void
bgr2rgb(xf::Mat<SRC_T, ROWS, COLS, NPC> & _src,xf::Mat<DST_T, ROWS, COLS,
NPC> & _dst)
```

```
template<int SRC_T,int DST_T,int ROWS,int COLS,int NPC=1>void
rgb2bgr(xf::Mat<SRC_T, ROWS, COLS, NPC> & _src,xf::Mat<DST_T, ROWS, COLS,
NPC> & _dst)
```

Parameter Descriptions

The following table describes the template and the function parameters.

Table 203: **Function Parameter Descriptions**

Parameter	Description
SRC_T	Input pixel type. Only 8-bit, unsigned, 3-channel is supported (XF_8UC3).
DST_T	Output pixel type. Only 8-bit, unsigned, 3-channel is supported (XF_8UC3).

Table 203: **Function Parameter Descriptions** *(cont'd)*

Parameter	Description
ROWS	Maximum height of input and output image.
COLS	Maximum width of input and output image. Must be multiple of N.
NPC	Number of Pixels to be processed per cycle. Possible options are XF_NPPC1,XF_NPPC2,XF_NPPC4 and XF_NPPC8.
_src	BGR/RGB input image
_dst	RGB/BGR output image

Resource Utilization

The following table summarizes the resource utilization of ***RGB to BGR/ BGR to RGB*** function in Normal mode (1-Pixel), as generated in the Vivado HLS 2019.1 tool for the Xilinx xczu9eg-ffvb1156-2-i-es2 FPGA.

Operating Mode	Operating Frequency (MHz)	Utilization Estimate				
		BRAM_18K	DSP_48Es	FF	LUT	CLB
1 pixel	300	0	0	317	118	98

Performance Estimate

The following table summarizes the performance of the kernel in single pixel configuration as generated using Vivado HLS 2019.1 tool for the Xilinx xczu9eg-ffvb1156-2-i-es2 FPGA to process a HD (1080x1920) image.

Table 204: **Performance Estimate Summary**

Operating Mode	Latency Estimate
	Max Latency (ms)
1 pixel operation (300 MHz)	6.9

RGB/BGR to UYVY/YUYV

The `RGB/BGR to UYVY/YUYV` function converts a 3- channel RGB/BGR image to a single-channel YUYV/UYVY (YUV 4:2:2) image format. YUYV is a sub-sampled format, 2 RGBA pixel gives set of YUYV/UYVY values. YUYV/UYVY is represented in 16-bit values whereas, RGB is represented in 24-bit values

API Syntax

RGB to UYVY:

```
template<int SRC_T,int DST_T,int ROWS,int COLS,int NPC=1>void
rgb2uyvy(xf::Mat<SRC_T, ROWS, COLS, NPC> & _src,xf::Mat<DST_T, ROWS, COLS,
NPC> & _dst)
```

RGB to YUYV:

```
template<int SRC_T,int DST_T,int ROWS,int COLS,int NPC=1>void
rgb2yuyv(xf::Mat<SRC_T, ROWS, COLS, NPC> & _src,xf::Mat<DST_T, ROWS, COLS,
NPC> & _dst)
```

BGR to UYVY:

```
template<int SRC_T,int DST_T,int ROWS,int COLS,int NPC=1>void
bgr2uyvy(xf::Mat<SRC_T, ROWS, COLS, NPC> & _src,xf::Mat<DST_T, ROWS, COLS,
NPC> & _dst)
```

BGR to YUYV

```
template<int SRC_T,int DST_T,int ROWS,int COLS,int NPC=1>void
bgr2yuyv(xf::Mat<SRC_T, ROWS, COLS, NPC> & _src,xf::Mat<DST_T, ROWS, COLS,
NPC> & _dst)
```

Parameter Descriptions

The following table describes the template and the function parameters.

Table 205: **Function Parameter Descriptions**

Parameter	Description
SRC_T	Input pixel type. Only 8-bit, unsigned, 3-channel is supported (XF_8UC3)
DST_T	Output pixel type. Only 16-bit, unsigned, 1-channel is supported (XF_16UC1)
ROWS	Maximum height of input and output image
COLS	Maximum width of input and output image. Must be multiple of NPC.
NPC	Number of pixels to be processed per cycle. Possible options are XF_NPPC1,XF_NPPC2,XF_NPPC4 and XF_NPPC8..
_src	RGB/BGR input image
_dst	UYVY/YUYV output image

Resource Utilization

The following table summarizes the resource utilization of `RGB/BGR` to `UYVY/YUYV` function in normal mode(1-Pixel), as generated in the Vivado HLS 2019.1 tool for the Xilinx xczu9eg-ffvb1156-2-i-es2 FPGA.

Operating Mode	Operating Frequency (MHz)	Utilization Estimate				
		BRAM_18K	DSP_48Es	FF	LUT	CLB
1 pixel	300	0	9	249	203	55

Performance Estimate

The following table summarizes the performance of the kernel in single pixel configuration as generated using Vivado HLS 2019.1 tool for the Xilinx xczu9eg-ffvb1156-2-i-es2 FPGA to process a HD (1080x1920) image.

Table 206: **Performance Estimate Summary**

Operating Mode	Latency Estimate
	Max Latency (ms)
1 pixel operation (300 MHz)	6.9

XYZ to RGB/BGR

The `xyz2rgb` function converts XYZ color space to 3-channel RGB image.

$$\begin{bmatrix} R \\ G \\ B \end{bmatrix} = \begin{bmatrix} 3.240479 & -1.53715 & -0.498535 \\ -0.969256 & 1.875991 & 0.041556 \\ 0.055648 & -0.204043 & 1.057311 \end{bmatrix} \cdot \begin{bmatrix} X \\ Y \\ Z \end{bmatrix}$$

API Syntax

```
template<int SRC_T,int DST_T,int ROWS,int COLS,int NPC=1>void
xyz2rgb(xf::Mat<SRC_T, ROWS, COLS, NPC> & _src,xf::Mat<DST_T, ROWS, COLS,
NPC> & _dst)template<int SRC_T,int DST_T,int ROWS,int COLS,int NPC=1>void
xyz2bgr(xf::Mat<SRC_T, ROWS, COLS, NPC> & _src,xf::Mat<DST_T, ROWS, COLS,
NPC> & _dst)
```

Parameter Descriptions

The following table describes the template and the function parameters.

Table 207: **XYZ2RGB/BGR Function Parameter Descriptions**

Parameter	Description
SRC_T	Input pixel type. Only 8-bit, unsigned, 3-channel is supported (XF_8UC3).
DST_T	Output pixel type. Only 8-bit, unsigned, 3-channel is supported (XF_8UC3).
ROWS	Maximum height of input and output image. Must be multiple of 8.
COLS	Maximum width of input and output image. Must be multiple of 8.
NPC	Number of pixels to be processed per cycle.
_src	XYZ input image.

Table 207: **XYZ2RGB/BGR Function Parameter Descriptions** *(cont'd)*

Parameter	Description
_dst	RGB/BGR output image.

Resource Utilization

The following table summarizes the resource utilization of XYZ2RGB/BGR for different configurations, as generated in the Vivado HLS 2019.1 version tool for the Xczu9eg-ffvb1156-1-i-es1 FPGA, to process a HD (1080x1920) image.

Table 208: **XYZ2RGB/BGR Function Resource Utilization Summary**

Operating Mode	Operating Frequency (MHz)	Utilization Estimate			
		BRAM_18K	DSP_48Es	FF	LUT
1 pixel	300	0	8	639	401

Performance Estimate

The following table summarizes the performance of XYZ2RGB/BGR for different configurations, as generated using the Vivado HLS 2019.1 version tool for the Xczu9eg-ffvb1156-1-i-es1, to process a HD (1080x1920) image.

Table 209: **XYZ2RGB/BGRFunction Performance Estimate Summary**

Operating Mode	Latency Estimate
	Max Latency (ms)
1 pixel operation (300 MHz)	6.9

Color Thresholding

The `colorthresholding` function compares the color space values of the source image with low and high threshold values, and returns either 255 or 0 as the output.

API Syntax

```
template<int SRC_T,int DST_T,int MAXCOLORS, int ROWS, int COLS,int NPC>
          void colorthresholding(xf::Mat<SRC_T, ROWS, COLS, NPC> &
_src_mat,xf::Mat<DST_T, ROWS, COLS, NPC> & _dst_mat,unsigned char
low_thresh[MAXCOLORS*3], unsigned char high_thresh[MAXCOLORS*3])
```

Parameter Descriptions

The table below describes the template and the function parameters.

Parameter	Description
SRC_T	Input pixel type. Only 8-bit, unsigned, 3 channel is supported (XF_8UC3).
DST_T	Output pixel type. Only 8-bit, unsigned, 1 channel is supported (XF_8UC1).
MAXCOLORS	Maximum number of color values
ROWS	Maximum height of input and output image
COLS	Maximum width of input and output image. Must be a multiple of 8, for 8 pixel mode.
NPC	Number of pixels to be processed per cycle. Only XF_NPPC1 supported.
_src_mat	Input image
_dst_mat	Thresholded image
low_thresh	Lowest threshold values for the colors
high_thresh	Highest threshold values for the colors

Compare

The Compare function performs the per element comparison of pixels in two corresponding images src1, src2 and stores the result in dst.

$$dst(x,y)=src1(x,y)\ CMP\_OP\ src2(x,y)$$

CMP_OP – a flag specifies correspondence between the pixels.

- XF_CMP_EQ : src1 is equal to src2
- XF_CMP_GT : src1 is greater than src2
- XF_CMP_GE : src1 is greater than or equal to src2
- XF_CMP_LT : src1 is less than src2
- XF_CMP_LE : src1 is less than or equal to src2
- XF_CMP_NE : src1 is unequal to src2

If the comparison result is true, then the corresponding element of dst is set to 255; else it is set to 0.

API Syntax

```
template<int CMP_OP,  int SRC_T , int ROWS, int COLS, int NPC=1>
void compare(xf::Mat<SRC_T, ROWS, COLS, NPC> & _src1, xf::Mat<SRC_T, ROWS,
COLS, NPC> & _src2, xf::Mat<SRC_T, ROWS, COLS, NPC> & _dst)
```

Parameter Descriptions

The following table describes the template and the function parameters.

Table 210: **Compare Function Parameter Descriptions**

Parameter	Description
CMP_OP	The flag that specify the relation between the elements needs to be checked
SRC_T	Input Pixel Type. 8-bit, unsigned, 1 channel is supported (XF_8UC1)
ROWS	Maximum height of input and output image.
COLS	Maximum width of input and output image. In case of N-pixel parallelism, width should be multiple of N
NPC	Number of pixels to be processed per cycle; possible options are XF_NPPC1 and XF_NPPC8 for 1 pixel and 8 pixel operations respectively.
_src1	First input image
_src2	Second input image
_dst	Output image

Resource Utilization

The following table summarizes the resource utilization of the Compare XF_CMP_NE configuration in Resource optimized (8 pixels) mode and normal mode as generated using Vivado HLS 2019.1 version tool for the Xczu9eg-ffvb1156-1-i-es1 FPGA.

Table 211: **Compare Function Resource Utilization Summary**

Name	Resource Utilization	
	1 pixel per clock operation	8 pixel per clock operation
	300 MHz	150 MHz
BRAM_18K	0	0
DSP48E	0	0
FF	87	60
LUT	38	84
CLB	16	20

Performance Estimate

The following table summarizes a performance estimate of the kernel in different configurations, generated using Vivado HLS 2019.1 tool for Xczu9eg-ffvb1156-1-i-es1 FPGA to process a grayscale HD (1080x1920) image.

Table 212: **Compare Function Performance Estimate Summary**

Operating Mode	Latency Estimate	
	Operating Frequency (MHz)	Latency (in ms)
1 pixel	300	6.9
8 pixel	150	1.7

CompareS

The CompareS function performs the comparison of a pixel in the input image (src1) and the given scalar value scl, and stores the result in dst.

$$dst(x,y)=src1(x,y)\ CMP\_OP\ scalar$$

CMP_OP – a flag specifies correspondence between the pixel and the scalar.

- XF_CMP_EQ : src1 is equal to scl
- XF_CMP_GT : src1 is greater than scl
- XF_CMP_GE : src1 is greater than or equal to scl
- XF_CMP_LT : src1 is less than scl
- XF_CMP_LE : src1 is less than or equal to scl
- XF_CMP_NE : src1 is unequal to scl

If the comparison result is true, then the corresponding element of dst is set to 255, else it is set to 0.

API Syntax

```
template<int CMP_OP,  int SRC_T , int ROWS, int COLS, int NPC=1>
void compareS(xf::Mat<SRC_T, ROWS, COLS, NPC> & _src1, unsigned char
_scl[XF_CHANNELS(SRC_T,NPC)], xf::Mat<SRC_T, ROWS, COLS, NPC> & _dst)
```

Parameter Descriptions

The following table describes the template and the function parameters.

Table 213: **CompareS Function Parameter Descriptions**

Parameter	Description
CMP_OP	The flag that specifying the relation between the elements to be checked
SRC_T	Input pixel type. 8-bit, unsigned, 1 channel is supported (XF_8UC1).
ROWS	Maximum height of input and output image
COLS	Maximum width of input and output image. In case of N-pixel parallelism, the width should be a multiple of N
NPC	Number of pixels to be processed per cycle; possible options are XF_NPPC1 and XF_NPPC8 for 1 pixel and 8 pixels operations respectively.
_src1	First input image
_scl	Input scalar value, the size should be number of channels
_dst	Output image

Resource Utilization

The following table summarizes the resource utilization of the CompareS function with XF_CMP_NE configuration in Resource optimized (8 pixels) mode and normal mode as generated using Vivado HLS 2019.1 version tool for the Xczu9eg-ffvb1156-1-i-es1 FPGA

Table 214: **CompareS Function Resource Utilization Summary**

Name	Resource Utilization	
	1 pixel per clock operation	8 pixel per clock operation
	300 MHz	150 MHz
BRAM_18K	0	0
DSP48E	0	0
FF	93	93
LUT	39	68
CLB	21	28

Performance Estimate

The following table summarizes a performance estimate of the kernel in different configurations, generated using Vivado HLS 2019.1 tool for Xczu9eg-ffvb1156-1-i-es1 FPGA to process a grayscale HD (1080x1920) image.

Table 215: **CompareS Function Performance Estimate Summary**

Operating Mode	Latency Estimate	
	Operating Frequency (MHz)	Latency (ms)
1 pixel	300	6.9
8 pixel	150	1.7

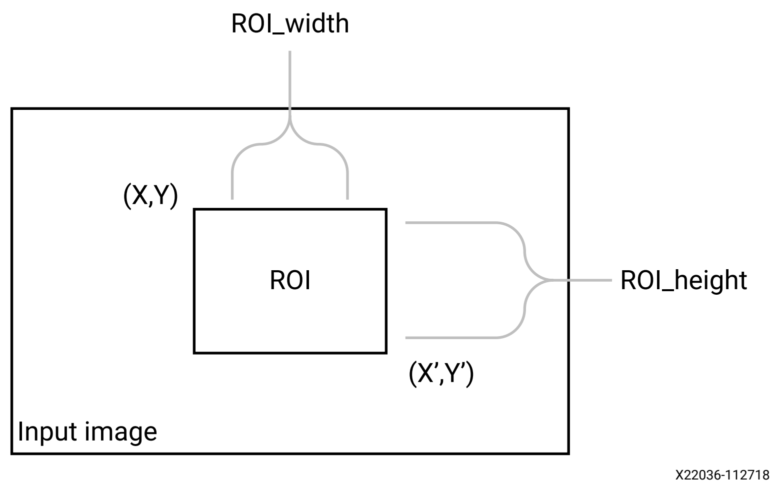
Crop

The `Crop` function extracts the region of interest (ROI) from the input image.

$$P(X,Y) \leq P(xi, yi) \leq P(X', Y')$$

- P(X,Y) - Top left corner of ROI
- P(X',Y') - Bottom Right of ROI

Figure 9: **Crop Function**



API Syntax

```
template<int SRC_T, int ROWS, int COLS,int ARCH_TYPE=0,int NPC=1>
void crop(xf::Mat<SRC_T, ROWS, COLS, NPC> & _src_mat,xf::Mat<SRC_T, ROWS,
COLS, NPC>  &_dst_mat,xf::Rect_<unsigned int> &roi)
```

Parameter Descriptions

The following table describes the template and the function parameters.

Table 216: **Crop Function Parameter Descriptions**

Parameter	Description
SRC_T	Input pixel type. Only 8-bit, unsigned, 1 and 3 channels are supported (XF_8UC1 and XF_8UC3).
ROWS	Maximum height of input and output image.
COLS	Maximum width of input and output image. Must be multiple of 8 for 8-pixel operation.
ARCH_TYPE	Architecture type. 0 resolves to stream implementation and 1 resolves to memory mapped implementation.
NPC	Number of pixels to be processed per cycle. NPC should be power of 2.
_src_mat	Input image
_dst_mat	Output ROI image
roi	ROI is a `xf::Rect` object that consists of the top left corner of the rectangle along with the height and width of the rectangle.

Resource Utilization

The following table summarizes the resource utilization of crop function in normal mode (NPC=1) for 3 ROIs (480x640, 100x200, 300x300) as generated in the Vivado HLS 2019.1 tool for the Xilinx xczu9eg-ffvb1156-2-i-es2 FPGA.

Table 217: **Crop Function Resource Utilization Summary**

Name	Resource Utilization	
	1-pixel per clock operation	8-pixel per clock operation
	300 MHz	300MHz
BRAM_18K	6	8
DSP48E	10	10
FF	17482	16995
LUT	16831	15305

Performance Estimate

The following table summarizes a performance estimate of the kernel in different configurations, generated using Vivado HLS 2019.1 tool for Xczu9eg-ffvb1156-1-i-es1 FPGA to process a grayscale HD (1080x1920) image for 3 ROIs (480x640, 100x200, 300x300).

Table 218: **Crop Function Performance Estimate Summary**

Operating Mode	Latency Estimate	
	Operating Frequency (MHz)	Latency (ms)
1 pixel	300	1.7
8 pixel	300	0.6

Multiple ROI Extraction

You can call the `xf::crop` function multiple times in `accel.cpp`.

Multiple ROI Extraction Example

```
void crop_accel(xf::Mat<TYPE, HEIGHT, WIDTH, NPIX>
&_src,xf::Mat<TYPE,HEIGHT, WIDTH, NPIX> _dst[NUM_ROI],xf::Rect_<unsigned
int> roi[NUM_ROI])
```

```
 {xf::crop<TYPE, TYPE, HEIGHT, WIDTH, NPIX>(_src, _dst[0],roi[0]);
xf::crop<TYPE, TYPE, HEIGHT, WIDTH, NPIX>(_src, _dst[1],roi[1]);
xf::crop<TYPE, TYPE, HEIGHT, WIDTH, NPIX>(_src, _dst[2],roi[2]);}
```

Custom Convolution

The `filter2D` function performs convolution over an image using a user-defined kernel.

Convolution is a mathematical operation on two functions *f* and *g*, producing a third function, The third function is typically viewed as a modified version of one of the original functions, that gives the area overlap between the two functions to an extent that one of the original functions is translated.

The filter can be unity gain filter or a non-unity gain filter. The filter must be of type XF_16SP. If the co-efficients are floating point, it must be converted into the Qm.n and provided as the input as well as the shift parameter has to be set with the ‘n’ value. Else, if the input is not of floating point, the filter is provided directly and the shift parameter is set to zero.

API Syntax

```
template<int BORDER_TYPE,int FILTER_WIDTH,int FILTER_HEIGHT, int SRC_T,int
DST_T, int ROWS, int COLS,int NPC=1>
void filter2D(xf::Mat<SRC_T, ROWS, COLS, NPC> & _src_mat,xf::Mat<DST_T,
ROWS, COLS, NPC> & _dst_mat,short int
filter[FILTER_HEIGHT*FILTER_WIDTH],unsigned char _shift)
```

Parameter Descriptions

The following table describes the template and the function parameters.

Table 219: **filter2D Function Parameter Descriptions**

Parameter	Description
BORDER_TYPE	Border Type supported is XF_BORDER_CONSTANT
FILTER_HEIGHT	Number of rows in the input filter
FILTER_WIDTH	Number of columns in the input filter
SRC_T	Input pixel type. Only 8-bit, unsigned, 1 and 3 channels are supported (XF_8UC1 and XF_8UC3)
DST_T	Output pixel type.8-bit unsigned single and 3 channels (XF_8UC1,XF_8UC3) and 16-bit signed single and 3 channels (XF_16SC1,XF_16SC3) supported.
ROWS	Maximum height of input and output image
COLS	Maximum width of input and output image. Must be multiple of 8, for 8 pixel mode.
NPC	Number of pixels to be processed per cycle; possible options are XF_NPPC1 and XF_NPPC8 for 1 pixel and 8 pixel operations respectively.
_src_mat	Input image
_dst_mat	Output image
filter	The input filter of any size, provided the dimensions should be an odd number. The filter co-efficients either a 16-bit value or a 16-bit fixed point equivalent value.
_shift	The filter must be of type XF_16SP. If the co-efficients are floating point, it must be converted into the Qm.n and provided as the input as well as the shift parameter has to be set with the ‘n’ value. Else, if the input is not of floating point, the filter is provided directly and the shift parameter is set to zero.

Resource Utilization

The following table summarizes the resource utilization of the kernel in different configurations, generated using Vivado HLS 2019.1 tool for the Xczu9eg-ffvb1156-1-i-es1 FPGA, to process a grayscale HD (1080x1920) image.

Table 220: **filter2D Function Resource Utilization Summary**

Operating Mode	Filter Size	Operating Frequency (MHz)	Utilization Estimate				
			BRAM_18K	DSP_48Es	FF	LUT	CLB
1 pixel	3x3	300	3	9	1701	1161	269
	5x5	300	5	25	3115	2144	524
8 pixel	3x3	150	6	72	2783	2768	638
	5x5	150	10	216	3020	4443	1007

The following table summarizes the resource utilization of the kernel in different configurations, generated using Vivado HLS 2019.1 tool for the Xilinx Xczu9eg-ffvb1156-1-i-es1 FPGA, to process a 4K 3 Channel image.

Table 221: **filter2D Function Resource Utilization Summary**

Operating Mode	Filter Size	Operating Frequency (MHz)	Utilization Estimate			
			BRAM_18K	DSP_48Es	FF	LUT
1 pixel	3x3	300	18	27	886	801
	5x5	300	30	75	1793	1445

Performance Estimate

The following table summarizes the performance of the kernel in different configurations, as generated using Vivado HLS 2019.1 tool for the Xczu9eg-ffvb1156-1-i-es1, to process a grayscale HD (1080x1920) image.

Table 222: **filter2D Function Performance Estimate Summary**

Operating Mode	Operating Frequency (MHz)	Filter Size	Latency Estimate
			Max (ms)
1 pixel	300	3x3	7
	300	5x5	7.1
8 pixel	150	3x3	1.86
	150	5x5	1.86

Delay

In image processing pipelines, it is possible that the inputs to a function with FIFO interfaces are not synchronized. That is, the first data packet for first input might arrive a finite number of clock cycles after the first data packet of the second input. If the function has FIFOs at its interface with insufficient depth, this causes the whole design to stall on hardware. To synchronize the inputs, we provide this function to delay the input packet that arrives early, by a finite number of clock cycles.

API Syntax

```
template<int MAXDELAY, int SRC_T, int ROWS, int COLS,int NPC=1 >
          void delayMat(xf::Mat<SRC_T, ROWS, COLS, NPC> & _src,
xf::Mat<SRC_T, ROWS, COLS, NPC> & _dst)
```

Parameter Descriptions

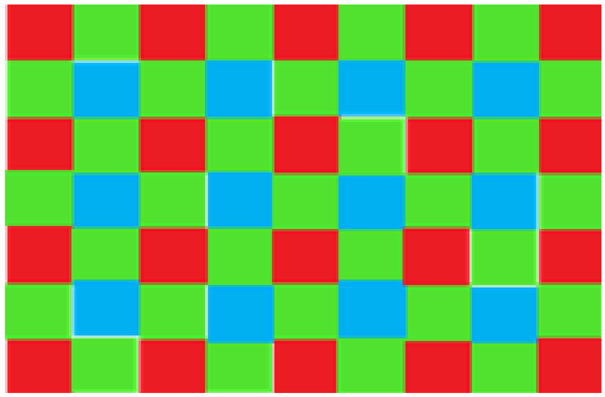
The table below describes the template and the function parameters.

Parameter	Description
SRC_T	Input and output pixel type
ROWS	Maximum height of input and output image.
COLS	Maximum width of input and output image (must be a multiple of 8, for 8 pixel operation)
NPC	Number of pixels to be processed per cycle; possible options are XF_NPPC1 and XF_NPPC8 for 1 pixel and 8 pixel operations respectively.
MAXDELAY	Maximum delay that the function is to be instantiated for.
_src	Input image
_dst	Output image

Demosaicing

The Demosaicing function converts a single plane Bayer pattern output, from the digital camera sensors to a color image. This function implements an improved bi-linear interpolation technique proposed by Malvar, He, and Cutler.

Figure 10: **Bayer Mosaic for Color Image**



The above figure shows the Bayer mosaic for color image capture in single-CCD digital cameras.

API Syntax

```
template<int BFORMAT, int SRC_T, int DST_T, int ROWS, int COLS, int
NPC,bool USE_URAM=false>
void demosaicing(xf::Mat<SRC_T, ROWS, COLS, NPC> &src_mat, xf::Mat<DST_T,
ROWS, COLS, NPC> &dst_mat)
```

Parameter Descriptions

The following table describes the template and the function parameters.

Table 223: **Demosaicing Function Parameter Descriptions**

Parameter	Description
BFORMAT	Input Bayer pattern. XF_BAYER_BG, XF_BAYER_GB, XF_BAYER_GR, and XF_BAYER_RG are the supported values.
SRC_T	Input pixel type. 8-bit, unsigned,1 and 3 channel (XF_8UC1 and XF_8UC3) and 16-bit, unsigned, 1 and 3 channel (XF_16UC1 and XF_16UC3) are supported.
DST_T	Output pixel type. 8-bit, unsigned, 4 channel (XF_8UC4) and 16-bit, unsigned, 4 channel (XF_16UC4) are supported.
ROWS	Number of rows in the image being processed.
COLS	Number of columns in the image being processed. Must be multiple of 8, in case of 8 pixel mode.
NPC	Number of pixels to be processed per cycle; single pixel parallelism (XF_NPPC1), two-pixel parallelism (XF_NPPC2) and four-pixel parallelism (XF_NPPC4) are supported. XF_NPPC4 is not supported with XF_16UC1 pixel type.
USE_URAM	Enable to map storage structures to UltraRAM.
_src_mat	Input image
_dst_mat	Output image

Resource Utilization

The following table below shows the resource utilization of the Demosaicing function, generated using Vivado HLS 2019.1 version tool for the Xczu9eg-ffvb1156-1-i-es1 FPGA.

Table 224: **Demosaicing Function Resource Utilization Summary**

Operating Mode	Operating Frequency (MHz)	Utilization Estimate				
		BRAM_18K	DSP48E	FF	LUT	CLB
1 pixel	300	8	0	1906	1915	412
2 pixel	300	8	0	2876	3209	627
4 pixel	300	8	0	2950	3222	660

The following table shows the resource utilization of the Demosaicing function, generated using SDx 2019.1 version tool for the xczu7ev-ffvc1156-2-e FPGA.

Table 225: **Demosaicing Function Resource Utilization Summary with UltraRAM Enabled**

Operating Mode	Operating Frequency (MHz)	Utilization Estimate					
		BRAM_18K	URAM	DSP48E	FF	LUT	CLB
1 pixel	300	0	1	0	1366	1339	412

Performance Estimate

The following table shows the performance in different configurations, generated using Vivado HLS 2019.1 tool for Xczu9eg-ffvb1156-1-i-es1 to process a 4K (3840x2160) image.

Table 226: **Demosaicing Function Performance Estimate Summary**

Operating Mode	Latency Estimate
	Max Latency (ms)
1 pixel operation (300 MHz)	27.82
2 pixel operation (300 MHz)	13.9
4 pixel operation (300 MHz, 8-bit image only)	6.95

Dilate

During a dilation operation, the current pixel intensity is replaced by the maximum value of the intensity in a nxn neighborhood of the current pixel.

$$dst(x, y) = \max_{\substack{x-1 \leq x' \leq x+1 \\ y-1 \leq y' \leq y+1}} src(x', y')$$

API Syntax

```
template<int BORDER_TYPE, int TYPE, int ROWS, int COLS,int K_SHAPE,int
K_ROWS,int K_COLS, int ITERATIONS, int NPC=1>
void dilate (xf::Mat<TYPE, ROWS, COLS, NPC> & _src, xf::Mat<TYPE, ROWS,
COLS, NPC> & _dst,unsigned char _kernel[K_ROWS*K_COLS])
```

Parameter Descriptions

The following table describes the template and the function parameters.

Table 227: **dilate Function Parameter Descriptions**

Parameter	Description
BORDER_TYPE	Border Type supported is XF_BORDER_CONSTANT
TYPE	Input and Output pixel type. Only 8-bit, unsigned, 1 and 3 channels are supported (XF_8UC1 and XF_8UC3)
ROWS	Maximum height of input and output image.
COLS	Maximum width of input and output image (must be multiple of 8, for 8-pixel operation)
K_SHAPE	Shape of the kernel . The supported kernel shapes are RECT, CROSS, and ELLIPSE.
NPC	Number of pixels to be processed per cycle; possible options are XF_NPPC1 and XF_NPPC8 for 1 pixel and 8 pixel operations respectively.
K_ROWS	Height of the kernel.
K_COLS	Width of the kernel.
ITERATIONS	Number of times the dilation is applied. Currently supporting for Rectangular shape kernel element.
_src_mat	Input image
_dst_mat	Output image
_kernel	Dilation kernel of size K_ROWS * K_COLS.

Resource Utilization

The following table summarizes the resource utilization of the Dilation function with rectangle shape structuring element in 1 pixel operation and 8 pixel operation, generated using Vivado HLS 2019.1 version tool for the Xczu9eg-ffvb1156-1-i-es1 FPGA for HD (1080X1920) image.

Table 228: **dilate Function Resource Utilization Summary**

Name	Resource Utilization	
	1 pixel per clock operation	8 pixel per clock operation
	300 MHz	150 MHz
BRAM_18K	3	6
DSP48E	0	0
FF	411	657
LUT	392	1249
CLB	96	255

The following table summarizes the resource utilization of the Dilation function with rectangle shape structuring element in 1 pixel operation, generated using Vivado HLS 2019.1 version tool for the Xczu9eg-ffvb1156-1-i-es1 FPGA for 4K 3channel image.

Table 229: **dilate Function Resource Utilization Summary**

Name	Resource Utilization
	1 pixel per clock operation
	300 MHz
BRAM_18K	18
DSP48E	0
FF	983
LUT	745
CLB	186

Performance Estimate

The following table summarizes a performance estimate of the Dilation function for Normal Operation (1 pixel) and Resource Optimized (8 pixel) configurations, generated using Vivado HLS 2019.1 tool for Xczu9eg-ffvb1156-1-i-es1 FPGA.

Table 230: **dilate Function Performance Estimate Summary**

Operating Mode	Latency Estimate	
	Min (ms)	Max (ms)
1 pixel (300 MHz)	7.0	7.0
8 pixel (150 MHz)	1.87	1.87

Duplicate

When various functions in a pipeline are implemented by a programmable logic, FIFOs are instantiated between two functions for dataflow processing. When the output from one function is consumed by two functions in a pipeline, the FIFOs need to be duplicated. This function facilitates the duplication process of the FIFOs.

API Syntax

```
template<int SRC_T, int ROWS, int COLS,int NPC=1>
          void duplicateMat(xf::Mat<SRC_T, ROWS, COLS, NPC> & _src,
xf::Mat<SRC_T, ROWS, COLS, NPC> & _dst1,xf::Mat<SRC_T, ROWS, COLS, NPC> &
_dst2)
```

Parameter Descriptions

The table below describes the template and the function parameters.

Parameter	Description
SRC_T	Input and output pixel type
ROWS	Maximum height of input and output image.
COLS	Maximum width of input and output image (must be a multiple of 8, for 8-pixel operation)
NPC	Number of pixels to be processed per cycle. Possible options are XF_NPPC1 and XF_NPPC8 for 1 pixel and 8 pixel operations respectively.
_src	Input image
_dst1	Duplicate output for _src
_dst2	Duplicate output for _src

Erode

The `erode` function finds the minimum pixel intensity in the NXN neighborhood of a pixel and replaces the pixel intensity with the minimum value.

$$dst(x, y) = \min_{\substack{x-1 \le x' \le x+1 \\ y-1 \le y' \le y+1}} src(x', y')$$

API Syntax

```
template<int BORDER_TYPE, int TYPE, int ROWS, int COLS,int K_SHAPE,int
K_ROWS,int K_COLS, int ITERATIONS, int NPC=1>
void erode (xf::Mat<TYPE, ROWS, COLS, NPC> & _src, xf::Mat<TYPE, ROWS,
COLS, NPC> & _dst,unsigned char _kernel[K_ROWS*K_COLS])
```

Parameter Descriptions

The following table describes the template and the function parameters.

Table 231: **erode Function Parameter Descriptions**

Parameter	Description
BORDER_TYPE	Border type supported is XF_BORDER_CONSTANT
TYPE	Input and Output pixel type. Only 8-bit, unsigned, 1 and 3 channels are supported (XF_8UC1 and XF_8UC3)
ROWS	Maximum height of input and output image.
COLS	Maximum width of input and output image (must be multiple of 8, for 8-pixel operation)
K_SHAPE	Shape of the kernel . The supported kernel shapes are RECT,CROSS and ELLIPSE.
K_ROWS	Height of the kernel.
K_COLS	Width of the kernel.
ITERATIONS	Number of times the erosion is applied.Currently supporting for Rectangular shape kernel element.
NPC	Number of pixels to be processed per cycle; possible options are XF_NPPC1 and XF_NPPC8 for 1 pixel and 8 pixel operations respectively.

Table 231: **erode Function Parameter Descriptions** *(cont'd)*

Parameter	Description
_src_mat	Input image
_dst_mat	Output image
_kernel	Erosion kernel of size K_ROWS * K_COLS.

Resource Utilization

The following table summarizes the resource utilization of the Erosion function with rectangular shape structuring element generated using Vivado HLS 2019.1 version tool for the Xczu9eg-ffvb1156-1-i-es1 FPGA,for FullHD image(1080x1920).

Table 232: **erode Function Resource Utilization Summary**

Name	Resource Utilization	
	1 pixel per clock operation	8 pixel per clock operation
	300 MHz	150 MHz
BRAM_18K	3	6
DSP48E	0	0
FF	411	657
LUT	392	1249
CLB	96	255

The following table summarizes the resource utilization of the Erosion function with rectangular shape structuring element generated using Vivado HLS 2019.1 version tool for the Xczu9eg-ffvb1156-1-i-es1 FPGA,for 4K image with 3channels.

Table 233: **erode Function Resource Utilization Summary**

Name	Resource Utilization
	1 pixel per clock operation
	300 MHz
BRAM_18K	18
DSP48E	0
FF	983
LUT	3745
CLB	186

Performance Estimate

The following table summarizes a performance estimate of the Erosion function for Normal Operation (1 pixel) and Resource Optimized (8 pixel) configurations, generated using Vivado HLS 2019.1 tool for Xczu9eg-ffvb1156-1-i-es1 FPGA.

Table 234: **erode Function Performance Estimate Summary**

Operating Mode	Latency Estimate	
	Min (ms)	Max (ms)
1 pixel (300 MHz)	7.0	7.0
8 pixel (150 MHz)	1.85	1.85

FAST Corner Detection

Features from accelerated segment test (FAST) is a corner detection algorithm, that is faster than most of the other feature detectors.

The `fast` function picks up a pixel in the image and compares the intensity of 16 pixels in its neighborhood on a circle, called the Bresenham's circle. If the intensity of 9 contiguous pixels is found to be either more than or less than that of the candidate pixel by a given threshold, then the pixel is declared as a corner. Once the corners are detected, the non-maximal suppression is applied to remove the weaker corners.

This function can be used for both still images and videos. The corners are marked in the image. If the corner is found in a particular location, that location is marked with 255, otherwise it is zero.

API Syntax

```
template<int NMS,int SRC_T,int ROWS, int COLS,int NPC=1>
void fast(xf::Mat<SRC_T, ROWS, COLS, NPC> & _src_mat,xf::Mat<SRC_T, ROWS,
COLS, NPC> & _dst_mat,unsigned char _threshold)
```

Parameter Descriptions

The following table describes the template and the function parameters.

Table 235: **fast Function Parameter Descriptions**

Parameter	Description
NMS	If NMS == 1, non-maximum suppression is applied to detected corners (keypoints). The value should be 0 or 1.
SRC_T	Input pixel type. Only 8-bit, unsigned, 1-channel is supported (XF_8UC1)
ROWS	Maximum height of input image.
COLS	Maximum width of input image (must be a multiple of 8, for 8-pixel operation)
NPC	Number of pixels to be processed per cycle; possible options are XF_NPPC1 and XF_NPPC8 for 1 pixel and 8 pixel operations respectively.
_src_mat	Input image
_dst_mat	Output image. The corners are marked in the image.
_threshold	Threshold on the intensity difference between the center pixel and its neighbors. Usually it is taken around 20.

Resource Utilization

The following table summarizes the resource utilization of the kernel for different configurations, generated using Vivado HLS 2019.1 for the Xczu9eg-ffvb1156-1-i-es1 FPGA, to process a grayscale HD (1080x1920) image with NMS.

Table 236: **fast Function Resource Utilization Summary**

Name	Resource Utilization	
	1 pixel	8 pixel
	300 MHz	150 MHz
BRAM_18K	10	20
DSP48E	0	0
FF	2695	7310
LUT	3792	20956
CLB	769	3519

Performance Estimate

The following table summarizes the performance of kernel for different configurations, as generated using Vivado HLS 2019.1 tool for the Xczu9eg-ffvb1156-1-i-es1, to process a grayscale HD (1080x1920) image with non-maximum suppression (NMS).

Table 237: **fast Function Performance Estimate Summary**

Operating Mode	Operating Frequency (MHz)	Filter Size	Latency Estimate
			Max (ms)
1 pixel	300	3x3	7
8 pixel	150	3x3	1.86

Gaussian Filter

The `GaussianBlur` function applies Gaussian blur on the input image. Gaussian filtering is done by convolving each point in the input image with a Gaussian kernel.

$$G_0(x, y) = e^{\frac{-(x-\mu_x)^2}{2\sigma_x^2} + \frac{-(y-\mu_y)^2}{2\sigma_y^2}}$$

Where μ_x, μ_y are the mean values and σ_x, σ_y are the variances in x and y directions respectively. In the GaussianBlur function, values of μ_x, μ_y are considered as zeroes and the values of σ_x, σ_y are equal.

API Syntax

```
template<int FILTER_SIZE, int BORDER_TYPE, int SRC_T, int ROWS, int COLS,
int NPC =  1>
void GaussianBlur(xf::Mat<SRC_T, ROWS, COLS, NPC> & src, xf::Mat<SRC_T,
ROWS, COLS, NPC> & dst, float sigma)
```

Parameter Descriptions

The following table describes the template and the function parameters.

Table 238: **GaussianBlur Function Parameter Descriptions**

Parameter	Description
FILTER_SIZE	Filter size. Filter size of 3 (XF_FILTER_3X3), 5 (XF_FILTER_5X5) and 7 (XF_FILTER_7X7) are supported.
BORDER_TYPE	Border type supported is XF_BORDER_CONSTANT
SRC_T	Input and Output pixel type. Only 8-bit, unsigned, 1 and 3 channels are supported (XF_8UC1 and XF_8UC3)
ROWS	Maximum height of input and output image.
COLS	Maximum width of input and output image (must be a multiple of 8, for 8-pixel operation)
NPC	Number of pixels to be processed per cycle; possible values are XF_NPPC1 and XF_NPPC8 for 1 pixel and 8 pixel operations respectively.
src	Input image
dst	Output image
sigma	Standard deviation of Gaussian filter

Resource Utilization

The following table summarizes the resource utilization of the Gaussian Filter in different configurations, generated using Vivado HLS 2019.1 version tool for the Xczu9eg-ffvb1156-1-i-es1 FPGA, to progress a grayscale HD (1080x1920) image.

Table 239: **GaussianBlur Function Resource Utilization Summary**

Operating Mode	Filter Size	Operating Frequency (MHz)	Utilization Estimate				
			BRAM_18K	DSP_48Es	FF	LUT	CLB
1 pixel	3x3	300	3	17	3641	2791	610
	5x5	300	5	27	4461	3544	764
	7x7	250	7	35	4770	4201	894
8 pixel	3x3	150	6	52	3939	3784	814
	5x5	150	10	111	5688	5639	1133
	7x7	150	14	175	7594	7278	1518

The following table summarizes the resource utilization of the Gaussian Filter in different configurations, generated using Vivado HLS 2019.1 version tool for the Xczu9eg-ffvb1156-1-i-es1 FPGA, to progress a 4K 3 Channel image.

Table 240: **GaussianBlur Function Resource Utilization Summary**

Operating Mode	Filter Size	Operating Frequency (MHz)	Utilization Estimate			
			BRAM_18K	DSP_48Es	FF	LUT
1 pixel	3x3	300	18	33	4835	3472
	5x5	300	30	51	5755	3994
	7x7	300	42	135	8086	5422

Performance Estimate

The following table summarizes a performance estimate of the Gaussian Filter in different configurations, as generated using Vivado HLS 2019.1 tool for Xczu9eg-ffvb1156-1-i-es1 FPGA, to process a grayscale HD (1080x1920) image.

Table 241: **GaussianBlur Function Performance Estimate Summary**

Operating Mode	Filter Size	Latency Estimate Max Latency (ms)
1 pixel operation (300 MHz)	3x3	7.01
	5x5	7.03
	7x7	7.06
8 pixel operation (150 MHz)	3x3	1.6
	5x5	1.7
	7x7	1.74

Gradient Magnitude

The `magnitude` function computes the magnitude for the images. The input images are x-gradient and y-gradient images of type 16S. The output image is of same type as the input image.

For L1NORM normalization, the magnitude computed image is the pixel-wise added image of absolute of x-gradient and y-gradient, as shown below:.

$$g = |g_x| + |g_y|$$

For L2NORM normalization, the magnitude computed image is as follows:

$$g = \sqrt{(g_x^2 + g_y^2)}$$

API Syntax

```
template< int NORM_TYPE ,int SRC_T,int DST_T, int ROWS, int COLS,int NPC=1>
void magnitude(xf::Mat<SRC_T, ROWS, COLS, NPC> & _src_matx,xf::Mat<DST_T,
ROWS, COLS, NPC> & _src_maty,xf::Mat<DST_T, ROWS, COLS, NPC> & _dst_mat)
```

Parameter Descriptions

The following table describes the template and the function parameters.

Table 242: **magnitude Function Parameter Descriptions**

Parameter	Description
NORM_TYPE	Normalization type can be either L1 or L2 norm. Values are XF_L1NORM or XF_L2NORM
SRC_T	Input pixel type. Only 16-bit, signed, 1 channel is supported (XF_16SC1)
DST_T	Output pixel type. Only 16-bit, signed,1 channel is supported (XF_16SC1)
ROWS	Maximum height of input and output image.
COLS	Maximum width of input and output image (must be multiple of 8, for 8-pixel operation)
NPC	Number of pixels to be processed per cycle; possible values are XF_NPPC1 and XF_NPPC8 for 1 pixel and 8 pixel operations respectively.
_src_matx	First input, x-gradient image.
_src_maty	Second input, y-gradient image.
_dst_mat	Output, magnitude computed image.

Resource Utilization

The following table summarizes the resource utilization of the kernel in different configurations, generated using Vivado HLS 2019.1 tool for the Xczu9eg-ffvb1156-1-i-es1 FPGA, to process a grayscale HD (1080x1920) image and for L2 normalization.

Table 243: **magnitude Function Resource Utilization Summary**

Name	Resource Utilization	
	1 pixel	8 pixel
	300 MHz	150 MHz
BRAM_18K	0	0
DSP48E	2	16
FF	707	2002
LUT	774	3666
CLB	172	737

Performance Estimate

The following table summarizes the performance of the kernel in different configurations, as generated using Vivado HLS 2019.1 tool for the Xczu9eg-ffvb1156-1-i-es1, to process a grayscale HD (1080x1920) image and for L2 normalization.

Table 244: **magnitude Function Performance Estimate Summary**

Operating Mode	Operating Frequency (MHz)	Latency Estimate
		Max (ms)
1 pixel	300	7.2
8 pixel	150	1.7

Gradient Phase

The `phase` function computes the polar angles of two images. The input images are x-gradient and y-gradient images of type 16S. The output image is of same type as the input image.

For radians:

$$angle(x,\ y) = atan2(g_y,\ \ g_x)$$

For degrees:

$$angle(x,\ y) = atan2(g_y,\ \ g_x)*\frac{180}{\pi}$$

API Syntax

```
template<int RET_TYPE ,int SRC_T,int DST_T, int ROWS, int COLS,int NPC=1 >
void phase(xf::Mat<SRC_T, ROWS, COLS, NPC> & _src_matx,xf::Mat<DST_T, ROWS,
COLS, NPC> & _src_maty,xf::Mat<DST_T, ROWS, COLS, NPC> & _dst_mat)
```

Parameter Descriptions

The following table describes the template and the function parameters.

Table 245: **phase Function Parameter Descriptions**

Parameter	Description
RET_TYPE	Output format can be either in radians or degrees. Options are XF_RADIANS or XF_DEGREES. • If the XF_RADIANS option is selected, phase API will return result in Q4.12 format. The output range is (0, 2 pi). • If the XF_DEGREES option is selected, xFphaseAPI will return result in Q10.6 degrees and output range is (0, 360).

Table 245: **phase Function Parameter Descriptions** *(cont'd)*

Parameter	Description
SRC_T	Input pixel type. Only 16-bit, signed, 1 channel is supported (XF_16SC1).
DST_T	Output pixel type. Only 16-bit, signed, 1 channel is supported (XF_16SC1)
ROWS	Maximum height of input and output image.
COLS	Maximum width of input and output image (must be a multiple of 8, for 8-pixel operation)
NPC	Number of pixels to be processed per cycle; possible options are XF_NPPC1 and XF_NPPC8 for 1 pixel and 8 pixel operations respectively.
_src_matx	First input, x-gradient image.
_src_maty	Second input, y-gradient image.
_dst_mat	Output, phase computed image.

Resource Utilization

The following table summarizes the resource utilization of the kernel in different configurations, generated using Vivado HLS 2019.1 tool for the Xczu9eg-ffvb1156-1-i-es1 FPGA, to process a grayscale HD (1080x1920) image.

Table 246: **phase Function Resource Utilization Summary**

Name	Resource Utilization	
	1 pixel	8 pixel
	300 MHz	150 MHz
BRAM_18K	6	24
DSP48E	6	19
FF	873	2396
LUT	753	3895
CLB	185	832

Performance Estimate

The following table summarizes the performance of the kernel in different configurations, as generated using Vivado HLS 2019.1 tool for the Xczu9eg-ffvb1156-1-i-es1, to process a grayscale HD (1080x1920) image.

Table 247: **phase Function Performance Estimate Summary**

Operating Mode	Operating Frequency (MHz)	Latency Estimate (ms)
1 pixel	300	7.2
8 pixel	150	1.7

Deviation from OpenCV

In phase implementation, the output is returned in a fixed point format. If XF_RADIANS option is selected, phase API will return result in Q4.12 format. The output range is (0, 2 pi). If XF_DEGREES option is selected, phase API will return result in Q10.6 degrees and output range is (0, 360).

Harris Corner Detection

In order to understand Harris Corner Detection, let us consider a grayscale image. Sweep a window `w(x,y)` (with displacements `u` in the x-direction and `v` in the y-direction), `I` calculates the variation of intensity `w(x,y)`.

$$E(u,\ v) = \sum\ w(x,\ y)[I(x+u,\ y+v) - I(x,\ y)]^2$$

Where:

- `w(x,y)` is the window position at (x,y)
- `I(x,y)` is the intensity at (x,y)
- `I(x+u,y+v)` is the intensity at the moved window `(x+u,y+v)`.

Since we are looking for windows with corners, we are looking for windows with a large variation in intensity. Hence, we have to maximize the equation above, specifically the term:

$$[I(x+u,\ y+v) - I(x,\ y)]^2$$

Using Taylor expansion:

$$E(u,\ v) = \sum\ [I(x,\ y) + uI_x + vI_y - I(x,\ y)]^2$$

Expanding the equation and cancelling `I(x,y)` with `-I(x,y)`:

$$E(u,\ v) = \sum\ u^2 I_x^2 + 2uvI_x I_y + v^2 I_y^2$$

The above equation can be expressed in a matrix form as:

$$E(u,\ v) = [u\ v]\left(\sum\ w(x,\ y)\begin{bmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{bmatrix}\right)\begin{bmatrix} u \\ v \end{bmatrix}$$

So, our equation is now:

$$E(u,\ v) = [u\ v]M\begin{bmatrix} u \\ v \end{bmatrix}$$

A score is calculated for each window, to determine if it can possibly contain a corner:

$$R = det(M) - k(trace(M))^2$$

Where,

- $det(M) = \lambda_1 \lambda_2$
- $trace(M) = \lambda_1 + \lambda_2$

Non-Maximum Suppression:

In non-maximum suppression (NMS) if radius = 1, then the bounding box is 2*r+1 = 3.

In this case, consider a 3x3 neighborhood across the center pixel. If the center pixel is greater than the surrounding pixel, then it is considered a corner. The comparison is made with the surrounding pixels, which are within the radius.

Radius = 1

x-1, y-1	x-1, y	x-1, y+1
x, y-1	x, y	x, y+1
x+1, y-1	x+1, y	x+1, y+1

Threshold:

A threshold=442, 3109 and 566 is used for 3x3, 5x5, and 7x7 filters respectively. This threshold is verified over 40 sets of images. The threshold can be varied, based on the application. The corners are marked in the output image. If the corner is found in a particular location, that location is marked with 255, otherwise it is zero.

API Syntax

```
template<int FILTERSIZE,int BLOCKWIDTH, int NMSRADIUS,int SRC_T,int ROWS,
int COLS,int NPC=1,bool USE_URAM=false>
void cornerHarris(xf::Mat<SRC_T, ROWS, COLS, NPC> & src,xf::Mat<SRC_T,
ROWS, COLS, NPC> & dst,uint16_t threshold, uint16_t k)
```

Parameter Descriptions

The following table describes the template and the function parameters.

Table 248: **cornerHarris Function Parameter Descriptions**

Parameter	Description
FILTERSIZE	Size of the Sobel filter. 3, 5, and 7 supported.
BLOCKWIDTH	Size of the box filter. 3, 5, and 7 supported.
NMSRADIUS	Radius considered for non-maximum suppression. Values supported are 1 and 2.

Table 248: **cornerHarris Function Parameter Descriptions** *(cont'd)*

Parameter	Description
TYPE	Input pixel type. Only 8-bit, unsigned, 1-channel is supported (XF_8UC1).
ROWS	Maximum height of input image.
COLS	Maximum width of input image (must be multiple of 8, for 8-pixel operation)
NPC	Number of pixels to be processed per cycle; possible options are XF_NPPC1 and XF_NPPC8 for 1 pixel and 8 pixel operations respectively.
USE_URAM	Enable to map some storage structures to URAM
src	Input image
dst	Output image.
threshold	Threshold applied to the corner measure.
k	Harris detector parameter

Resource Utilization

The following table summarizes the resource utilization of the Harris corner detection in different configurations, generated using Vivado HLS 2019.1 version tool for the Xczu9eg-ffvb1156-1-i-es1 FPGA, to process a grayscale HD (1080x1920) image.

The following table summarizes the resource utilization for Sobel Filter = 3, Box filter=3 and NMS_RADIUS =1.

Table 249: **Resource Utilization Summary - For Sobel Filter = 3, Box filter=3 and NMS_RADIUS =1**

Name	Resource Utilization	
	1 pixel	8 pixel
	300 MHz	150 MHz
BRAM_18K	33	66
DSP48E	10	80
FF	3254	9330
LUT	3522	13222
CLB	731	2568

The following table summarizes the resource utilization for Sobel Filter = 3, Box filter=5 and NMS_RADIUS =1.

Table 250: **Resource Utilization Summary - Sobel Filter = 3, Box filter=5 and NMS_RADIUS =1**

Name	Resource Utilization	
	1 pixel	8 pixel
	300 MHz	150 MHz
BRAM_18K	45	90

Table 250: **Resource Utilization Summary - Sobel Filter = 3, Box filter=5 and NMS_RADIUS =1** *(cont'd)*

Name	Resource Utilization	
	1 pixel	8 pixel
	300 MHz	150 MHz
DSP48E	10	80
FF	5455	12459
LUT	5675	24594
CLB	1132	4498

The following table summarizes the resource utilization for Sobel Filter = 3, Box filter=7 and NMS_RADIUS =1.

Table 251: **Resource Utilization Summary - Sobel Filter = 3, Box filter=7 and NMS_RADIUS =1**

Name	Resource Utilization	
	1 pixel	8 pixel
	300 MHz	150 MHz
BRAM_18K	57	114
DSP48E	10	80
FF	8783	16593
LUT	9157	39813
CLB	1757	6809

The following table summarizes the resource utilization for Sobel Filter = 5, Box filter=3 and NMS_RADIUS =1.

Table 252: **Resource Utilization Summary - Sobel Filter = 5, Box filter=3 and NMS_RADIUS =1**

Name	Resource Utilization	
	1 pixel	8 pixel
	300 MHz	200 MHz
BRAM_18K	35	70
DSP48E	10	80
FF	4656	11659
LUT	4681	17394
CLB	1005	3277

The following table summarizes the resource utilization for Sobel Filter = 5, Box filter=5 and NMS_RADIUS =1.

Table 253: **Resource Utilization Summary - Sobel Filter = 5, Box filter=5 and NMS_RADIUS =1**

Name	Resource Utilization	
	1 pixel	8 pixel
	300 MHz	150 MHz
BRAM_18K	47	94
DSP48E	10	80
FF	6019	14776
LUT	6337	28795
CLB	1353	5102

The following table summarizes the resource utilization for Sobel Filter = 5, Box filter=7 and NMS_RADIUS =1.

Table 254: **Resource Utilization Summary - Sobel Filter = 5, Box filter=7 and NMS_RADIUS =1**

Name	Resource Utilization	
	1 pixel	8 pixel
	300 MHz	150 MHz
BRAM_18K	59	118
DSP48E	10	80
FF	9388	18913
LUT	9414	43070
CLB	1947	7508

The following table summarizes the resource utilization for Sobel Filter = 7, Box filter=3 and NMS_RADIUS =1.

Table 255: **Resource Utilization Summary - Sobel Filter = 7, Box filter=3 and NMS_RADIUS =1**

Name	Resource Utilization	
	1 pixel	8 pixel
	300 MHz	150 MHz
BRAM_18K	37	74
DSP48E	11	88
FF	6002	13880
LUT	6337	25573
CLB	1327	4868

The following table summarizes the resource utilization for Sobel Filter = 7, Box filter=5 and NMS_RADIUS =1.

Table 256: **Resource Utilization Summary - Sobel Filter = 7, Box filter=5 and NMS_RADIUS =1**

Name	Resource Utilization	
	1 pixel	8 pixel
	300 MHz	150 MHz
BRAM_18K	49	98
DSP48E	11	88
FF	7410	17049
LUT	8076	36509
CLB	1627	6518

The following table summarizes the resource utilization for Sobel Filter = 7, Box filter=7 and NMS_RADIUS =1.

Table 257: **Resource Utilization Summary - Sobel Filter = 7, Box filter=7 and NMS_RADIUS =1**

Name	Resource Utilization	
	1 pixel	8 pixel
	300 MHz	150 MHz
BRAM_18K	61	122
DSP48E	11	88
FF	10714	21137
LUT	11500	51331
CLB	2261	8863

The following table summarizes the resource utilization for Sobel Filter = 3, Box filter=3 and NMS_RADIUS =2.

Table 258: **Resource Utilization Summary - Sobel Filter = 3, Box filter=3 and NMS_RADIUS =2**

Name	Resource Utilization	
	1 pixel	8 pixel
	300 MHz	150 MHz
BRAM_18K	41	82
DSP48E	10	80
FF	5519	10714
LUT	5094	16930
CLB	1076	3127

The following table summarizes the resource utilization for Sobel Filter = 3, Box filter=5 and NMS_RADIUS =2.

Table 259: **Resource Utilization Summary**

Name	Resource Utilization	
	1 pixel	8 pixel
	300 MHz	150 MHz
BRAM_18K	53	106
DSP48E	10	80
FF	6798	13844
LUT	6866	28286
CLB	1383	4965

The following table summarizes the resource utilization for Sobel Filter = 3, Box filter=7 and NMS_RADIUS =2.

Table 260: **Resource Utilization Summary - Sobel Filter = 3, Box filter=7 and NMS_RADIUS =2**

Name	Resource Utilization	
	1 pixel	8 pixel
	300 MHz	150 MHz
BRAM_18K	65	130
DSP48E	10	80
FF	10137	17977
LUT	10366	43589
CLB	1940	7440

The following table summarizes the resource utilization for Sobel Filter = 5, Box filter=3 and NMS_RADIUS =2.

Table 261: **Resource Utilization Summary - Sobel Filter = 5, Box filter=3 and NMS_RADIUS =2**

Name	Resource Utilization	
	1 pixel	8 pixel
	300 MHz	150 MHz
BRAM_18K	43	86
DSP48E	10	80
FF	5957	12930
LUT	5987	21187
CLB	1244	3922

The following table summarizes the resource utilization for Sobel Filter = 5, Box filter=5 and NMS_RADIUS =2.

Table 262: **Resource Utilization Summary - Sobel Filter = 5, Box filter=5 and NMS_RADIUS =2**

Name	Resource Utilization	
	1 pixel	8 pixel
	300 MHz	150 MHz
BRAM_18K	55	110
DSP48E	10	80
FF	5442	16053
LUT	6561	32377
CLB	1374	5871

The following table summarizes the resource utilization for Sobel Filter = 5, Box filter=7 and NMS_RADIUS =2.

Table 263: **Resource Utilization Summary - Sobel Filter = 5, Box filter=7 and NMS_RADIUS =2**

Name	Resource Utilization	
	1 pixel	8 pixel
	300 MHz	150 MHz
BRAM_18K	67	134
DSP48E	10	80
FF	10673	20190
LUT	10793	46785
CLB	2260	8013

The following table summarizes the resource utilization for Sobel Filter = 7, Box filter=3 and NMS_RADIUS =2.

Table 264: **Resource Utilization Summary - Sobel Filter = 7, Box filter=3 and NMS_RADIUS =2**

Name	Resource Utilization	
	1 pixel	8 pixel
	300 MHz	150 MHz
BRAM_18K	45	90
DSP48E	11	88
FF	7341	15161
LUT	7631	29185
CLB	1557	5425

The following table summarizes the resource utilization for Sobel Filter = 7, Box filter=5 and NMS_RADIUS =2.

Table 265: **Resource Utilization Summary - Sobel Filter = 7, Box filter=5 and NMS_RADIUS =2**

Name	Resource Utilization	
	1 pixel	8 pixel
	300 MHz	150 MHz
BRAM_18K	57	114
DSP48E	11	88
FF	8763	18330
LUT	9368	40116
CLB	1857	7362

The following table summarizes the resource utilization for Sobel Filter = 7, Box filter=7 and NMS_RADIUS =2.

Table 266: **Resource Utilization Summary - Sobel Filter = 7, Box filter=7 and NMS_RADIUS =2**

Name	Resource Utilization	
	1 pixel	8 pixel
	300 MHz	150 MHz
BRAM_18K	69	138
DSP48E	11	88
FF	12078	22414
LUT	12831	54652
CLB	2499	9628

Resource Utilization with URAM enable

The following table summarizes the resource utilization of the Harris corner detection in different configurations, generated using SDx 2019.1 version tool for the xczu7ev-ffvc1156-2-e FPGA, to process a grayscale 4K (3840X2160) image.

The following table summarizes the resource utilization for Sobel Filter = 3, Box filter=3 and NMS_RADIUS =1.

Table 267: **Resource Utilization Summary - For Sobel Filter = 3, Box filter=3 and NMS_RADIUS =1**

Name	Resource Utilization	
	1 pixel	8 pixel
	300 MHz	150 MHz
BRAM_18K	12	12
URAM	4	21

Table 267: **Resource Utilization Summary - For Sobel Filter = 3, Box filter=3 and NMS_RADIUS =1** *(cont'd)*

Name	Resource Utilization	
	1 pixel	8 pixel
	300 MHz	150 MHz
DSP48E	10	80
FF	5306	11846
LUT	3696	13846

The following table summarizes the resource utilization for Sobel Filter = 3, Box filter=5 and NMS_RADIUS =1.

Table 268: **Resource Utilization Summary - Sobel Filter = 3, Box filter=5 and NMS_RADIUS =1**

Name	Resource Utilization	
	1 pixel	8 pixel
	300 MHz	150 MHz
BRAM_18K	12	12
URAM	7	30
DSP48E	10	80
FF	7625	13899
LUT	5596	27136

The following table summarizes the resource utilization for Sobel Filter = 3, Box filter=7 and NMS_RADIUS =1.

Table 269: **Resource Utilization Summary - Sobel Filter = 3, Box filter=7 and NMS_RADIUS =1**

Name	Resource Utilization	
	1 pixel	8 pixel
	300 MHz	150 MHz
BRAM_18K	12	12
URAM	7	42
DSP48E	10	80
FF	12563	19919
LUT	8816	39087

The following table summarizes the resource utilization for Sobel Filter = 5, Box filter=3 and NMS_RADIUS =1.

Table 270: **Resource Utilization Summary - Sobel Filter = 5, Box filter=3 and NMS_RADIUS =1**

Name	Resource Utilization	
	1 pixel	8 pixel
	300 MHz	150 MHz
BRAM_18K	12	12
URAM	4	23
DSP48E	10	80
FF	6689	15022
LUT	4506	18719

The following table summarizes the resource utilization for Sobel Filter = 5, Box filter=5 and NMS_RADIUS =1.

Table 271: **Resource Utilization Summary - Sobel Filter = 5, Box filter=5 and NMS_RADIUS =1**

Name	Resource Utilization	
	1 pixel	8 pixel
	300 MHz	150 MHz
BRAM_18K	12	12
URAM	7	32
DSP48E	10	80
FF	9050	17063
LUT	6405	31992

The following table summarizes the resource utilization for Sobel Filter = 5, Box filter=7 and NMS_RADIUS =1.

Table 272: **Resource Utilization Summary - Sobel Filter = 5, Box filter=7 and NMS_RADIUS =1**

Name	Resource Utilization	
	1 pixel	8 pixel
	300 MHz	150 MHz
BRAM_18K	12	12
URAM	7	44
DSP48E	10	80
FF	13946	23116
LUT	9626	44738

The following table summarizes the resource utilization for Sobel Filter = 7, Box filter=3 and NMS_RADIUS =1.

Table 273: **Resource Utilization Summary - Sobel Filter = 7, Box filter=3 and NMS_RADIUS =1**

Name	Resource Utilization	
	1 pixel	8 pixel
	300 MHz	150 MHz
BRAM_18K	12	12
URAM	4	25
DSP48E	11	88
FF	8338	17378
LUT	6151	24844

The following table summarizes the resource utilization for Sobel Filter = 7, Box filter=5 and NMS_RADIUS =1.

Table 274: **Resource Utilization Summary - Sobel Filter = 7, Box filter=5 and NMS_RADIUS =1**

Name	Resource Utilization	
	1 pixel	8 pixel
	300 MHz	150 MHz
BRAM_18K	12	12
URAM	7	34
DSP48E	11	88
FF	10497	19457
LUT	7858	39762

The following table summarizes the resource utilization for Sobel Filter = 7, Box filter=7 and NMS_RADIUS =1.

Table 275: **Resource Utilization Summary - Sobel Filter = 7, Box filter=7 and NMS_RADIUS =1**

Name	Resource Utilization	
	1 pixel	8 pixel
	300 MHz	150 MHz
BRAM_18K	12	12
URAM	7	46
DSP48E	11	88
FF	15393	25450
LUT	11080	50662

The following table summarizes the resource utilization for Sobel Filter = 3, Box filter=3 and NMS_RADIUS =2.

Table 276: **Resource Utilization Summary - Sobel Filter = 3, Box filter=3 and NMS_RADIUS =2**

Name	Resource Utilization	
	1 pixel	8 pixel
	300 MHz	150 MHz
BRAM_18K	20	20
URAM	4	21
DSP48E	10	80
FF	6286	13441
LUT	4704	18072

The following table summarizes the resource utilization for Sobel Filter = 3, Box filter=5 and NMS_RADIUS =2.

Table 277: **Resource Utilization Summary**

Name	Resource Utilization	
	1 pixel	8 pixel
	300 MHz	150 MHz
BRAM_18K	20	20
URAM	7	30
DSP48E	10	80
FF	8626	15498
LUT	6606	31371

The following table summarizes the resource utilization for Sobel Filter = 3, Box filter=7 and NMS_RADIUS =2.

Table 278: **Resource Utilization Summary - Sobel Filter = 3, Box filter=7 and NMS_RADIUS =2**

Name	Resource Utilization	
	1 pixel	8 pixel
	300 MHz	150 MHz
BRAM_18K	20	20
URAM	7	42
DSP48E	10	80
FF	13543	21522
LUT	9853	43301

The following table summarizes the resource utilization for Sobel Filter = 5, Box filter=3 and NMS_RADIUS =2.

Table 279: **Resource Utilization Summary - Sobel Filter = 5, Box filter=3 and NMS_RADIUS =2**

Name	Resource Utilization	
	1 pixel	8 pixel
	300 MHz	150 MHz
BRAM_18K	20	20
URAM	4	23
DSP48E	10	80
FF	7670	16750
LUT	5513	22854

The following table summarizes the resource utilization for Sobel Filter = 5, Box filter=5 and NMS_RADIUS =2.

Table 280: **Resource Utilization Summary - Sobel Filter = 5, Box filter=5 and NMS_RADIUS =2**

Name	Resource Utilization	
	1 pixel	8 pixel
	300 MHz	150 MHz
BRAM_18K	20	20
URAM	7	32
DSP48E	10	80
FF	9712	18793
LUT	7338	36136

The following table summarizes the resource utilization for Sobel Filter = 5, Box filter=7 and NMS_RADIUS =2.

Table 281: **Resource Utilization Summary - Sobel Filter = 5, Box filter=7 and NMS_RADIUS =2**

Name	Resource Utilization	
	1 pixel	8 pixel
	300 MHz	150 MHz
BRAM_18K	20	20
URAM	7	44
DSP48E	10	80
FF	14650	24846
LUT	10558	48866

The following table summarizes the resource utilization for Sobel Filter = 7, Box filter=3 and NMS_RADIUS =2.

Table 282: **Resource Utilization Summary - Sobel Filter = 7, Box filter=3 and NMS_RADIUS =2**

Name	Resource Utilization	
	1 pixel	8 pixel
	300 MHz	150 MHz
BRAM_18K	20	20
URAM	4	25
DSP48E	11	88
FF	9562	19101
LUT	7405	29986

The following table summarizes the resource utilization for Sobel Filter = 7, Box filter=5 and NMS_RADIUS =2.

Table 283: **Resource Utilization Summary - Sobel Filter = 7, Box filter=5 and NMS_RADIUS =2**

Name	Resource Utilization	
	1 pixel	8 pixel
	300 MHz	150 MHz
BRAM_18K	20	20
URAM	7	34
DSP48E	11	88
FF	11751	21180
LUT	9254	44024

The following table summarizes the resource utilization for Sobel Filter = 7, Box filter=7 and NMS_RADIUS =2.

Table 284: **Resource Utilization Summary - Sobel Filter = 7, Box filter=7 and NMS_RADIUS =2**

Name	Resource Utilization	
	1 pixel	8 pixel
	300 MHz	150 MHz
BRAM_18K	20	20
URAM	7	46
DSP48E	11	88
FF	16723	27156
LUT	12474	54858

Performance Estimate

The following table summarizes a performance estimate of the Harris corner detection in different configurations, as generated using Vivado HLS 2019.1 tool for Xilinx Xczu9eg-ffvb1156-1-i-es1 FPGA, to process a grayscale HD (1080x1920) image.

Table 285: **cornerHarris Function Performance Estimate Summary**

Operating Mode	Operating Frequency (MHz)	Configuration			Latency Estimate
		Sobel	Box	NMS Radius	Latency(In ms)
1 pixel	300 MHz	3	3	1	7
1 pixel	300 MHz	3	5	1	7.1
1 pixel	300 MHz	3	7	1	7.1
1 pixel	300 MHz	5	3	1	7.2
1 pixel	300 MHz	5	5	1	7.2
1 pixel	300 MHz	5	7	1	7.2
1 pixel	300 MHz	7	3	1	7.22
1 pixel	300 MHz	7	5	1	7.22
1 pixel	300 MHz	7	7	1	7.22
8 pixel	150 MHz	3	3	1	1.7
8 pixel	150 MHz	3	5	1	1.7
8 pixel	150 MHz	3	7	1	1.7
8 pixel	150 MHz	5	3	1	1.71
8 pixel	150 MHz	5	5	1	1.71
8 pixel	150 MHz	5	7	1	1.71
8 pixel	150 MHz	7	3	1	1.8
8 pixel	150 MHz	7	5	1	1.8
8 pixel	150 MHz	7	7	1	1.8
1 pixel	300 MHz	3	3	2	7.1
1 pixel	300 MHz	3	5	2	7.1
1 pixel	300 MHz	3	7	2	7.1
1 pixel	300 MHz	5	3	2	7.21
1 pixel	300 MHz	5	5	2	7.21
1 pixel	300 MHz	5	7	2	7.21
1 pixel	300 MHz	7	3	2	7.22
1 pixel	300 MHz	7	5	2	7.22
1 pixel	300 MHz	7	7	2	7.22
8 pixel	150 MHz	3	3	2	1.8
8 pixel	150 MHz	3	5	2	1.8
8 pixel	150 MHz	3	7	2	1.8
8 pixel	150 MHz	5	3	2	1.81

Table 285: **cornerHarris Function Performance Estimate Summary** *(cont'd)*

Operating Mode	Operating Frequency (MHz)	Configuration			Latency Estimate
		Sobel	Box	NMS Radius	Latency(In ms)
8 pixel	150 MHz	5	5	2	1.81
8 pixel	150 MHz	5	7	2	1.81
8 pixel	150 MHz	7	3	2	1.9
8 pixel	150 MHz	7	5	2	1.91
8 pixel	150 MHz	7	7	2	1.92

Deviation from OpenCV

In xfOpenCV thresholding and NMS are included, but in OpenCV they are not included. In xfOpenCV, all the blocks are implemented in fixed point. Whereas, in OpenCV, all the blocks are implemented in floating point.

Histogram Computation

The `calcHist` function computes the histogram of given input image.

$$H[src(x,\ y)] = H[src(x,\ y)] + 1$$

Where, H is the array of 256 elements.

API Syntax

```
template<int SRC_T,int ROWS, int COLS,int NPC=1>
void calcHist(xf::Mat<SRC_T, ROWS, COLS, NPC> & _src, uint32_t *histogram)
```

Parameter Descriptions

The following table describes the template and the function parameters.

Table 286: **calcHist Function Parameter Descriptions**

Parameter	Description
SRC_T	Input pixel type. Only 8-bit, unsigned, 1 and 3 channels are supported (XF_8UC1 and XF_8UC3)
ROWS	Maximum height of input and output image.
COLS	Maximum width of input and output image (must be multiple of 8, for 8-pixel operation)
NPC	Number of pixels to be processed per cycle
_src	Input image
histogram	Output array of 256 elements

Resource Utilization

The following table summarizes the resource utilization of the calcHist function for Normal Operation (1 pixel) and Resource Optimized (8 pixel) configurations, generated using Vivado HLS 2019.1 version tool for the Xczu9eg-ffvb1156-1-i-es1 FPGA at 300 MHz for 1 pixel case and at 150 MHz for 8 pixel mode.

Table 287: **calcHist Function Resource Utilization Summary**

Name	Resource Utilization	
	Normal Operation (1 pixel)	Resource Optimized (8 pixel)
BRAM_18K	2	16
DSP48E	0	0
FF	196	274
LUT	240	912
CLB	57	231

The following table summarizes the resource utilization of the calcHist function for Normal Operation (1 pixel), generated using Vivado HLS 2019.1 version tool for the Xczu9eg-ffvb1156-1-i-es1 FPGA at 300 MHz for 1 pixel case for 4K image 3channel .

Table 288: **calcHist Function Resource Utilization Summary**

Name	Resource Utilization
	Normal Operation (1 pixel)
BRAM_18K	8
DSP48E	0
FF	381
LUT	614
CLB	134

Performance Estimate

The following table summarizes a performance estimate of the calcHist function for Normal Operation (1 pixel) and Resource Optimized (8 pixel) configurations, generated using Vivado HLS 2019.1 version tool for the Xczu9eg-ffvb1156-1-i-es1 FPGA at 300 MHz for 1 pixel and 150 MHz for 8 pixel mode.

Table 289: **calcHist Function Performance Estimate Summary**

Operating Mode	Latency Estimate
	Max (ms)
1 pixel	6.9

Table 289: **calcHist Function Performance Estimate Summary** *(cont'd)*

Operating Mode	Latency Estimate
	Max (ms)
8 pixel	1.7

Histogram Equalization

The `equalizeHist` function performs histogram equalization on input image or video. It improves the contrast in the image, to stretch out the intensity range. This function maps one distribution (histogram) to another distribution (a wider and more uniform distribution of intensity values), so the intensities are spread over the whole range.

For histogram H[i], the cumulative distribution H'[i] is given as:

$$H'[i] = \sum_{0 \le j < i} H[j]$$

The intensities in the equalized image are computed as:

$$dst(x, y) = H'(src(x, y))$$

API Syntax

```
template<int SRC_T, int ROWS, int COLS, int NPC = 1>
void equalizeHist(xf::Mat<SRC_T, ROWS, COLS, NPC> & _src,xf::Mat<SRC_T,
ROWS, COLS, NPC> & _src1,xf::Mat<SRC_T, ROWS, COLS, NPC> & _dst)
```

Parameter Descriptions

The following table describes the template and the function parameters.

Table 290: **equalizeHist Function Parameter Descriptions**

Parameter	Description
SRC_T	Input and output pixel type. Only 8-bit, unsigned, 1 channel is supported (XF_8UC1)
ROWS	Maximum height of input and output image.
COLS	Maximum width of input and output image (must be a multiple of 8, for 8-pixel operation)
NPC	Number of pixels to be processed per cycle
_src	Input image
_src1	Input image
_dst	Output image

Resource Utilization

The following table summarizes the resource utilization of the equalizeHist function for Normal Operation (1 pixel) and Resource Optimized (8 pixel) configurations, generated using Vivado HLS 2019.1 version tool for the Xczu9eg-ffvb1156-1-i-es1 FPGA at 300 MHz for 1 pixel and 150 MHz for 8 pixel mode.

Table 291: **equalizeHist Function Resource Utilization Summary**

Operating Mode	Operating Frequency (MHz)	Utilization Estimate				
		BRAM_18K	DSP_48Es	FF	LUT	CLB
1 pixel	300	4	5	3492	1807	666
8 pixel	150	25	5	3526	2645	835

Performance Estimate

The following table summarizes a performance estimate of the equalizeHist function for Normal Operation (1 pixel) and Resource Optimized (8 pixel) configurations, generated using Vivado HLS 2019.1version tool for the Xczu9eg-ffvb1156-1-i-es1 FPGA at 300 MHz for 1 pixel and 150 MHz for 8 pixel mode.

Table 292: **equalizeHist Function Performance Estimate Summary**

Operating Mode	Latency Estimate
	Max (ms)
1 pixel per clock operation	13.8
8 pixel per clock operation	3.4

HOG

The histogram of oriented gradients (HOG) is a feature descriptor used in computer vision for the purpose of object detection. The feature descriptors produced from this approach is widely used in the pedestrian detection.

The technique counts the occurrences of gradient orientation in localized portions of an image. HOG is computed over a dense grid of uniformly spaced cells and normalized over overlapping blocks, for improved accuracy. The concept behind HOG is that the object appearance and shape within an image can be described by the distribution of intensity gradients or edge direction.

Both RGB and gray inputs are accepted to the function. In the RGB mode, gradients are computed for each plane separately, but the one with the higher magnitude is selected. With the configurations provided, the window dimensions are 64x128, block dimensions are 16x16.

API Syntax

```
template<int WIN_HEIGHT, int WIN_WIDTH, int WIN_STRIDE, int BLOCK_HEIGHT,
int BLOCK_WIDTH, int CELL_HEIGHT, int CELL_WIDTH, int NOB, int DESC_SIZE,
int IMG_COLOR, int OUTPUT_VARIANT, int SRC_T, int DST_T, int ROWS, int
COLS, int NPC = XF_NPPC1,bool USE_URAM=false>
void HOGDescriptor(xf::Mat<SRC_T, ROWS, COLS, NPC> &_in_mat, xf::Mat<DST_T,
1, DESC_SIZE, NPC> &_desc_mat);
```

Parameter Descriptions

The following table describes the template parameters.

Table 293: **HOGDescriptor Template Parameter Descriptions**

Parameters	Description
WIN_HEIGHT	The number of pixel rows in the window. This must be a multiple of 8 and should not exceed the number of image rows.
WIN_WIDTH	The number of pixel cols in the window. This must be a multiple of 8 and should not exceed the number of image columns.
WIN_STRIDE	The pixel stride between two adjacent windows. It is fixed at 8.
BLOCK_HEIGHT	Height of the block. It is fixed at 16.
BLOCK_WIDTH	Width of the block. It is fixed at 16.
CELL_HEIGHT	Number of rows in a cell. It is fixed at 8.
CELL_WIDTH	Number of cols in a cell. It is fixed at 8.
NOB	Number of histogram bins for a cell. It is fixed at 9
DESC_SIZE	The size of the output descriptor.
IMG_COLOR	The type of the image, set as either XF_GRAY or XF_RGB
OUTPUT_VARIENT	Must be either XF_HOG_RB or XF_HOG_NRB
SRC_T	Input pixel type. Must be either XF_8UC1 or XF_8UC4, for gray and color respectively.
DST_T	Output descriptor type. Must be XF_32UC1.
ROWS	Number of rows in the image being processed.
COLS	Number of columns in the image being processed.
NPC	Number of pixels to be processed per cycle; this function supports only XF_NPPC1 or 1 pixel per cycle operations.
USE_URAM	Enable to map UltraRAM instead of BRAM for some storage structures.

The following table describes the function parameters.

Table 294: **HOGDescriptor Function Parameter Descriptions**

Parameters	Description
_in_mat	Input image, of xf::Mat type
_desc_mat	Output descriptors, of xf::Mat type

Where,

- NO is normal operation (single pixel processing)
- RB is repetitive blocks (descriptor data are written window wise)
- NRB is non-repetitive blocks (descriptor data are written block wise, in order to reduce the number of writes).

***Note*:** In the RB mode, the block data is written to the memory taking the overlap windows into consideration. In the NRB mode, the block data is written directly to the output stream without consideration of the window overlap. In the host side, the overlap must be taken care.

Resource Utilization

The following table shows the resource utilization of `HOGDescriptor` function for normal operation (1 pixel) mode as generated in Vivado HLS 2019.1 version tool for the part Xczu9eg-ffvb1156-1-i-es1 at 300 MHz to process an image of 1920x1080 resolution.

Table 295: **HOGDescriptor Function Resource Utilization Summary**

Resource	Utilization (at 300 MHz) of 1 pixel operation			
	NRB		RB	
	Gray	RGB	Gray	RGB
BRAM_18K	43	49	171	177
DSP48E	34	46	36	48
FF	15365	15823	15205	15663
LUT	12868	13267	13443	13848

The following table shows the resource utilization of `HOGDescriptor` function for normal operation (1 pixel) mode as generated in SDx 2019.1 version tool for the part xczu7ev-ffvc1156-2-e at 300 MHz to process an image of 1920x1080 resolution with UltraRAM enabled.

Table 296: **HOGDescriptor Function Resource Utilization Summary with UltraRAM enabled**

Resource	Utilization (at 300 MHz) of 1 pixel operation			
	NRB		RB	
	Gray	RGB	Gray	RGB
BRAM_18K	10	12	18	20
URAM	15	15	15	17
DSP48E	34	46	36	48
FF	17285	17917	18270	18871
LUT	12409	12861	12793	13961

Performance Estimate

The following table shows the performance estimates of HOGDescriptor() function for different configurations as generated in Vivado HLS 2019.1 version tool for the part Xczu9eg-ffvb1156-1-i-es1 to process an image of 1920x1080p resolution.

Table 297: **HOGDescriptor Function Performance Estimate Summary**

Operating Mode	Operating Frequency (MHz)	Latency Estimate	
		Min (ms)	Max (ms)
NRB-Gray	300	6.98	8.83
NRB-RGBA	300	6.98	8.83
RB-Gray	300	176.81	177
RB-RGBA	300	176.81	177

Deviations from OpenCV

Listed below are the deviations from the OpenCV:

1. Border care

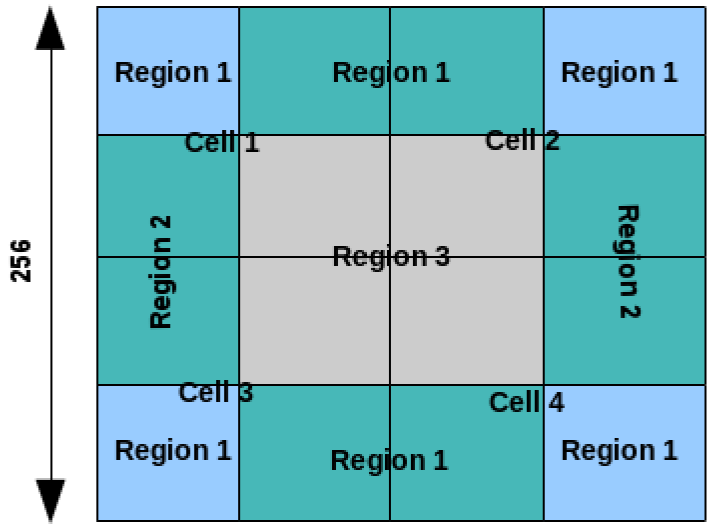
   The border care that OpenCV has taken in the gradient computation is BORDER_REFLECT_101, in which the border padding will be the neighboring pixels' reflection. Whereas, in the Xilinx implementation, BORDER_CONSTANT (zero padding) was used for the border care.

2. Gaussian weighing

   The Gaussian weights are multiplied on the pixels over the block, that is a block has 256 pixels, and each position of the block are multiplied with its corresponding Gaussian weights. Whereas, in the HLS implementation, gaussian weighing was not performed.

3. Cell-wise interpolation

   The magnitude values of the pixels are distributed across different cells in the blocks but on the corresponding bins.



Pixels in the region 1 belong only to its corresponding cells, but the pixels in region 2 and 3 are interpolated to the adjacent 2 cells and 4 cells respectively. This operation was not performed in the HLS implementation.

4. Output handling

   The output of the OpenCV will be in the column major form. In the HLS implementation, output will be in the row major form. Also, the feature vector will be in the fixed point type Q0.16 in the HLS implementation, while in the OpenCV it will be in floating point.

Limitations

1. The configurations are limited to Dalal's implementation
2. Image height and image width must be a multiple of cell height and cell width respectively.

HoughLines

The `HoughLines` function here is equivalent to HoughLines Standard in OpenCV. The `HoughLines` function is used to detect straight lines in a binary image. To apply the Hough transform, edge detection preprocessing is required. The input to the Hough transform is an edge detected binary image. For each point (xi,yi) in a binary image, we define a family of lines that go through the point as:

```
rho= xi cos(theta) + yi sin(theta)
```

[1] N. Dalal, B. Triggs: *Histograms of oriented gradients for human detection*, IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 2005.

Each pair of (rho,theta) represents a line that passes through the point (xi,yi). These (rho,theta) pairs of this family of lines passing through the point form a sinusoidal curve in (rho,theta) plane. If the sinusoids of N different points intersect in the (rho,theta) plane, then that intersection (rho1, theta1) represents the line that passes through these N points. In the `HoughLines` function, an accumulator is used to keep the count (also called voting) of all the intersection points in the (rho,theta) plane. After voting, the function filters spurious lines by performing thinning, that is, checking if the center vote value is greater than the neighborhood votes and threshold, then making that center vote as valid and other wise making it zero. Finally, the function returns the desired maximum number of lines (LINESMAX) in (rho,theta) form as output.

The design assumes the origin at the center of the image i.e at (Floor(COLS/2), Floor(ROWS/2)). The ranges of rho and theta are:

```
 theta = [0, pi)
```

```
rho=[-DIAG/2, DIAG/2), where DIAG = cvRound{SquareRoot( (COLS*COLS) +
(ROWS*ROWS))}
```

For ease of use, the input angles THETA, MINTHETA and MAXTHETA are taken in degrees, while the output theta is in radians. The angle resolution THETA is declared as an integer, but treated as a value in Q6.1 format (that is, THETA=3 signifies that the resolution used in the function is 1.5 degrees). When the output (rho, ϴ theta) is used for drawing lines, you should be aware of the fact that origin is at the center of the image.

API Syntax

```
template<unsigned int RHO,unsigned int THETA,int MAXLINES,int DIAG,int
MINTHETA,int MAXTHETA,int SRC_T, int ROWS, int COLS,int NPC>
```

```
void HoughLines(xf::Mat<SRC_T, ROWS, COLS, NPC> & _src_mat,float
outputrho[MAXLINES],float outputtheta[MAXLINES],short threshold,short
linesmax)
```

Parameter Descriptions

The following table describes the template and the function parameters.

Table 298: **HoughLines Function Parameter Descriptions**

Parameter	Description
RHO	Distance resolution of the accumulator in pixels.
THETA	Angle resolution of the accumulator in degrees and Q6.1 format.
MAXLINES	Maximum number of lines to be detected
MINTHETA	Minimum angle in degrees to check lines.
MAXTHETA	Maximum angle in degrees to check lines
DIAG	Diagonal of the image. It should be cvRound(sqrt(rows*rows + cols*cols)/RHO).

Table 298: **HoughLines Function Parameter Descriptions** *(cont'd)*

Parameter	Description
SRC_T	Input Pixel Type. Only 8-bit, unsigned, 1-channel is supported (XF_8UC1).
ROWS	Maximum height of input image
COLS	Maximum width of input image
NPC	Number of Pixels to be processed per cycle; Only single pixel supported XF_NPPC1.
_src_mat	Input image should be 8-bit, single-channel binary image.
outputrho	Output array of rho values. rho is the distance from the coordinate origin (center of the image).
outputtheta	Output array of theta values. Theta is the line rotation angle in radians.
threshold	Accumulator threshold parameter. Only those lines are returned that get enough votes (>threshold).
linesmax	Maximum number of lines.

Resource Utilization

The table below shows the resource utilization of the kernel for different configurations, generated using Vivado HLS 2019.1 version tool for the Xczu9eg-ffvb1156-1-i-es1 to process a grayscale HD (1080x1920) image for 512 lines.

Table 299: **Houghlines Function Resource Utilization Summary**

Name	Resource Utilization
	THETA=1, RHO=1
BRAM_18K	542
DSP48E	10
FF	60648
LUT	56131

Performance Estimate

The following table shows the performance of kernel for different configurations, generated using Vivado HLS 2019.1 version tool for the Xczu9eg-ffvb1156-1-i-es1 to process a grayscale HD (1080x1920) image for 512 lines.

Table 300: **Houghlines Function Performance Estimate Summary**

Operating Mode	Operating Frequency (MHz)	Latency Estimate
		Max (ms)
THETA=1, RHO=1	300	12.5

Pyramid Up

The `pyrUp` function is an image up-sampling algorithm. It first inserts zero rows and zero columns after every input row and column making up to the size of the output image. The output image size is always $(2*rows \times 2*columns)$. The zero padded image is then smoothened using Gaussian image filter. Gaussian filter for the pyramid-up function uses a fixed filter kernel as given below:

$$\frac{1}{256}\begin{bmatrix} 1 & 4 & 6 & 4 & 1 \\ 4 & 16 & 24 & 16 & 4 \\ 6 & 24 & 36 & 24 & 6 \\ 4 & 16 & 24 & 16 & 4 \\ 1 & 4 & 6 & 4 & 1 \end{bmatrix}$$

However, to make up for the pixel intensity that is reduced due to zero padding, each output pixel is multiplied by 4.

API Syntax

```
template<int TYPE, int ROWS, int COLS, int NPC>
void pyrUp (xf::Mat<TYPE, ROWS, COLS, NPC> & _src, xf::Mat<TYPE, ROWS,
COLS, NPC> & _dst)
```

Parameter Descriptions

The following table describes the template and the function parameters.

Table 301: **pyrUp Function Parameter Descriptions**

Parameter	Description
TYPE	Input and Output pixel type. Only 8-bit, unsigned, 1 and 3 channels are supported (XF_8UC1 and XF_8UC3)
ROWS	Maximum Height or number of output rows to build the hardware for this kernel
COLS	Maximum Width or number of output columns to build the hardware for this kernel
NPC	Number of pixels to process per cycle. Currently, the kernel supports only 1 pixel per cycle processing (XF_NPPC1).
_src	Input image stream
_dst	Output image stream

Resource Utilization

The following table summarizes the resource utilization of pyrUp for 1 pixel per cycle implementation, for a maximum input image size of 1920x1080 pixels. The results are after synthesis in Vivado HLS 2019.1 for the Xilinx Xczu9eg-ffvb1156-1-i-es1 FPGA at 300 MHz.

Table 302: **pyrUp Function Resource Utilization Summary**

Operating Mode	Operating Frequency (MHz)	Utilization Estimate			
		LUTs	FFs	DSPs	BRAMs
1 Pixel	300	1124	1199	0	10

The following table summarizes the resource utilization of pyrUp for 1 pixel per cycle implementation, for a maximum input image size of 4K with BGR. The results are after synthesis in Vivado HLS 2019.1 for the Xilinx Xczu9eg-ffvb1156-1-i-es1 FPGA at 300 MHz.

Table 303: **pyrUp Function Resource Utilization Summary**

Operating Mode	Operating Frequency (MHz)	Utilization Estimate			
		LUTs	FFs	DSPs	BRAMs
1 Pixel	300	2074	2176	0	59

Performance Estimate

The following table summarizes performance estimates of pyrUp function on Vivado HLS 2019.1 for the Xilinx Xczu9eg-ffvb1156-1-i-es1 FPGA.

Table 304: **pyrUp Function Performance Estimate Summary**

Operating Mode	Operating Frequency (MHz)	Input Image Size	Latency Estimate
			Max (ms)
1 pixel	300	1920x1080	27.82

Pyramid Down

The `pyrDown` function is an image down-sampling algorithm which smoothens the image before down-scaling it. The image is smoothened using a Gaussian filter with the following kernel:

$$\frac{1}{256}\begin{bmatrix} 1 & 4 & 6 & 4 & 1 \\ 4 & 16 & 24 & 16 & 4 \\ 6 & 24 & 36 & 24 & 6 \\ 4 & 16 & 24 & 16 & 4 \\ 1 & 4 & 6 & 4 & 1 \end{bmatrix}$$

Down-scaling is performed by dropping pixels in the even rows and the even columns. The resulting image size is $\left(\frac{rows+1}{2} \quad \frac{columns+1}{2}\right)$.

API Syntax

```
template<int TYPE, int ROWS, int COLS, int NPC,bool USE_URAM=false>
void pyrDown (xf::Mat<TYPE, ROWS, COLS, NPC> & _src, xf::Mat<TYPE, ROWS,
COLS, NPC> & _dst)
```

Parameter Descriptions

The following table describes the template and the function parameters.

Table 305: **pyrDown Function Parameter Descriptions**

Parameter	Description
TYPE	Input and Output pixel type. Only 8-bit, unsigned, 1 and 3 channels are supported (XF_8UC1 and XF_8UC3)
ROWS	Maximum Height or number of input rows to build the hardware for this kernel
COLS	Maximum Width or number of input columns to build the hardware for this kernel
NPC	Number of pixels to process per cycle. Currently, the kernel supports only 1 pixel per cycle processing (XF_NPPC1).
USE_URAM	Enable to map storage structures to UltraRAM
_src	Input image stream
_dst	Output image stream

Resource Utilization

The following table summarizes the resource utilization of pyrDown for 1 pixel per cycle implementation, for a maximum input image size of 1920x1080 pixels. The results are after synthesis in Vivado HLS 2019.1 for the Xilinx Xczu9eg-ffvb1156-1-i-es1 FPGA at 300 MHz.

Table 306: **pyrDown Function Resource Utilization Summary**

Operating Mode	Operating Frequency (MHz)	Utilization Estimate			
		LUTs	FFs	DSPs	BRAMs
1 Pixel	300	1171	1238	1	5

The following table summarizes the resource utilization of pyrDown for 1 pixel per cycle implementation, for a maximum input image size of 4Kwith BGR image. The results are after synthesis in Vivado HLS 2019.1 for the Xilinx Xczu9eg-ffvb1156-1-i-es1 FPGA at 300 MHz.

Table 307: **pyrDown Function Resource Utilization Summary**

Operating Mode	Operating Frequency (MHz)	Utilization Estimate			
		LUTs	FFs	DSPs	BRAMs
1 Pixel	300	2158	1983	2	30

The following table summarizes the resource utilization of pyrDown for 1 pixel per cycle implementation, for a maximum input image size of 3840x2160 pixels. The results are after synthesis in SDx 2019.1 for the Xilinx xczu7eg-ffvb1156-1 FPGA at 300 MHz with UltraRAM enabled.

Table 308: **pyrDown Function Resource Utilization Summary with UltraRAM Enabled**

Operating Mode	Operating Frequency (MHz)	Utilization Estimate				
		LUTs	FFs	DSPs	BRAMs	URAM
1 Pixel	300	1171	1243	0	0	1

Performance Estimate

The following table summarizes performance estimates of pyrDown function in Vivado HLS 2019.1 for the Xilinx Xczu9eg-ffvb1156-1-i-es1 FPGA.

Table 309: **pyrDown Function Performance Estimate Summary**

Operating Mode	Operating Frequency (MHz)	Input Image Size	Latency Estimate
			Max (ms)
1 pixel	300	1920x1080	6.99

InitUndistortRectifyMapInverse

The `InitUndistortRectifyMapInverse` function generates mapx and mapy, based on a set of camera parameters, where mapx and mapy are inputs for the xf::remap function. That is, for each pixel in the location (u, v) in the destination (corrected and rectified) image, the function computes the corresponding coordinates in the source image (the original image from camera). The InitUndistortRectifyMapInverse module is optimized for hardware, so the inverse of rotation matrix is computed outside the synthesizable logic. Note that the inputs are fixed point, so the floating point camera parameters must be type casted to Q12.20 format.

API Syntax

```
template< int CM_SIZE, int DC_SIZE, int MAP_T, int ROWS, int COLS, int NPC >
void InitUndistortRectifyMapInverse ( ap_fixed<32,12> *cameraMatrix,
ap_fixed<32,12> *distCoeffs, ap_fixed<32,12> *ir, xf::Mat<MAP_T, ROWS,
COLS, NPC> &_mapx_mat, xf::Mat<MAP_T, ROWS, COLS, NPC> &_mapy_mat, int
_cm_size, int _dc_size)
```

Parameter Descriptions

The following table describes the template and the function parameters.

Table 310: **InitUndistortRectifyMapInverse Function Parameter Descriptions**

Parameter	Description
CM_SIZE	It must be set at the compile time, 9 for 3x3 matrix
DC_SIZE	It must be set at the compile time, must be 4,5 or 8
MAP_T	It is the type of output maps, and must be XF_32FC1
ROWS	Maximum image height, necessary to generate the output maps
COLS	Maximum image width, necessary to generate the output maps
NPC	Number of pixels per cycle. This function supports only one pixel per cycle, so set to XF_NPPC1
cameraMatrix	The input matrix representing the camera in the old coordinate system
distCoeffs	The input distortion coefficients (k1,k2,p1,p2[,k3[,k4,k5,k6]])
ir	The input transformation matrix is equal to Invert(newCameraMatrix*R), where newCameraMatrix represents the camera in the new coordinate system and R is the rotation matrix.. This processing will be done outside the synthesizable block
_mapx_mat	Output mat objects containing the mapx
_mapy_mat	Output mat objects containing the mapy
_cm_size	9 for 3x3 matrix
_dc_size	4, 5 or 8. If this is 0, then it means there is no distortion

InRange

The InRange function checks if pixels in the image src lie between the given boundaries. dst(x,y) is set to 255, if src(x,y) is within the specified thresholds and otherwise 0.

$$Dst(I) = lowerb \leq src(I) \leq upperb$$

Where (x,y) is the spatial coordinate of the pixel.

API Syntax

```
template<int SRC_T, int ROWS, int COLS,int NPC=1>
void inRange(xf::Mat<SRC_T, ROWS, COLS, NPC> & src,unsigned char
lower_thresh,unsigned char upper_thresh,xf::Mat<SRC_T, ROWS, COLS, NPC> &
dst)
```

Parameter Descriptions

The following table describes the template and the function parameters.

Table 311: **InRange Function Parameter Descriptions**

Parameter	Description
SRC_T	Input Pixel Type. 8-bit, unsigned, 1 and 3 channels are supported (XF_8UC1 and XF_8UC3).
ROWS	Maximum height of input and output image.
COLS	Maximum width of input and output image. In case of N-pixel parallelism, width should be multiple of N.
NPC	Number of pixels to be processed per cycle; possible options are XF_NPPC1 and XF_NPPC8 for 1 pixel and 8 pixel operations respectively.
src	Input image
dst	Output image
lower_thresh	Lower threshold value
upper_thresh	Upper threshold value

Resource Utilization

The following table summarizes the resource utilization of the InRange function in Resource optimized (8 pixel) mode and normal mode as generated using Vivado HLS 2019.1 version tool for the Xczu9eg-ffvb1156-1-i-es1 FPGA

Table 312: **InRange Function Resource Utilization Summary**

Name	Resource Utilization	
	1 pixel per clock operation	8 pixel per clock operation
	300 MHz	150 MHz
BRAM_18K	0	0
DSP48E	0	0
FF	86	154
LUT	60	148
CLB	15	37

Performance Estimate

The following table summarizes a performance estimate of the kernel in different configurations, generated using Vivado HLS 2019.1 tool for Xczu9eg-ffvb1156-1-i-es1 FPGA to process a grayscale HD (1080x1920) image.

Table 313: **InRange Function Performance Estimate Summary**

Operating Mode	Latency Estimate	
	Operating Frequency (MHz)	Latency (ms)
1 pixel	300	6.9

Table 313: **InRange Function Performance Estimate Summary** *(cont'd)*

Operating Mode	Latency Estimate	
	Operating Frequency (MHz)	Latency (ms)
8 pixel	150	1.7

Integral Image

The `integral` function computes an integral image of the input. Each output pixel is the sum of all pixels above and to the left of itself.

$$dst(x,\ y) = sum(x,\ y) = sum(x,\ y) + sum(x - 1,\ y) + sum(x,\ y - 1) - sum(x - 1,\ y - 1)$$

API Syntax

```
template<int SRC_TYPE,int DST_TYPE, int ROWS, int COLS, int NPC=1>
void integral(xf::Mat<SRC_TYPE, ROWS, COLS, NPC> & _src_mat,
xf::Mat<DST_TYPE, ROWS, COLS, NPC> & _dst_mat)
```

Parameter Descriptions

The following table describes the template and the function parameters.

Table 314: **integral Function Parameter Descriptions**

Parameter	Description
SRC_TYPE	Input pixel type. Only 8-bit, unsigned, 1 channel is supported (XF_8UC1)
DST_TYPE	Output pixel type. Only 32-bit,unsigned,1 channel is supported(XF_32UC1)
ROWS	Maximum height of input and output image.
COLS	Maximum width of input and output image
NPC	Number of pixels to be processed per cycle; this function supports only XF_NPPC1 or 1 pixel per cycle operations.
_src_mat	Input image
_dst_mat	Output image

Resource Utilization

The following table summarizes the resource utilization of the kernel in different configurations, generated using Vivado HLS 2019.1 tool for the Xilinx Xczu9eg-ffvb1156-1-i-es1 FPGA, to process a grayscale HD (1080x1920) image.

Table 315: **integral Function Resource Utilization Summary**

Name	Resource Utilization
	1 pixel
	300 MHz
BRAM_18K	4
DSP48E	0
FF	613
LUT	378
CLB	102

Performance Estimate

The following table summarizes the performance of the kernel in different configurations, as generated using Vivado HLS 2019.1 tool for the Xilinx Xczu9eg-ffvb1156-1-i-es1, to process a grayscale HD (1080x1920) image.

Table 316: **integral Function Performance Estimate Summary**

Operating Mode	Latency Estimate	
	Operating Frequency (MHz)	Latency(in ms)
1pixel	300	7.2

Dense Pyramidal LK Optical Flow

Optical flow is the pattern of apparent motion of image objects between two consecutive frames, caused by the movement of object or camera. It is a 2D vector field, where each vector is a displacement vector showing the movement of points from first frame to second.

Optical Flow works on the following assumptions:

- Pixel intensities of an object do not have too many variations in consecutive frames
- Neighboring pixels have similar motion

Consider a pixel I(x, y, t) in first frame. (Note that a new dimension, time, is added here. When working with images only, there is no need of time). The pixel moves by distance (dx, dy) in the next frame taken after time dt. Thus, since those pixels are the same and the intensity does not change, the following is true:

$$I(x,\ y,\ t) = I(x + dx,\ y + dy,\ t + dt)$$

Taking the Taylor series approximation on the right-hand side, removing common terms, and dividing by dt gives the following equation:

$$f_x u + f_y v + f_t = 0$$

Where $f_x = \frac{\delta f}{\delta x}$, $f_y = \frac{\delta f}{\delta x}$, $u = \frac{dx}{dt}$ and $v = \frac{dy}{dt}$.

The above equation is called the Optical Flow equation, where, f_x and f_y are the image gradientsand f_t is the gradient along time. However, (u, v) is unknown. It is not possible to solve this equation with two unknown variables. Thus, several methods are provided to solve this problem. One method is Lucas-Kanade. Previously it was assumed that all neighboring pixels have similar motion. The Lucas-Kanade method takes a patch around the point, whose size can be defined through the 'WINDOW_SIZE' template parameter. Thus, all the points in that patch have the same motion. It is possible to find (f_x, f_y, f_t) for these points. Thus, the problem now becomes solving 'WINDOW_SIZE * WINDOW_SIZE' equations with two unknown variables,which is over-determined. A better solution is obtained with the "least square fit" method. Below is the final solution, which is a problem with two equations and two unknowns:

$$\begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} \sum f_{x_i}^2 & \sum f_{x_i} f_{y_i} \\ \sum f_{x_i} f_{y_i} & \sum f_{y_i}^2 \end{bmatrix}^{-1} \begin{bmatrix} -\sum f_{x_i} f_{t_i} \\ -\sum f_{y_i} f_{t_i} \end{bmatrix}$$

This solution fails when a large motion is involved and so pyramids are used. Going up in the pyramid, small motions are removed and large motions become small motions and so by applying Lucas-Kanade, the optical flow along with the scale is obtained.

API Syntax

```
template< int NUM_PYR_LEVELS, int NUM_LINES, int WINSIZE, int FLOW_WIDTH,
int FLOW_INT, int TYPE, int ROWS, int COLS, int NPC,bool USE_URAM=false>
void densePyrOpticalFlow(
xf::Mat<TYPE,ROWS,COLS,NPC> & _current_img,
xf::Mat<TYPE,ROWS,COLS,NPC> & _next_image,
xf::Mat<XF_32UC1,ROWS,COLS,NPC> & _streamFlowin,
xf::Mat<XF_32UC1,ROWS,COLS,NPC> & _streamFlowout,
const int level, const unsigned char scale_up_flag, float scale_in,
ap_uint<1> init_flag)
```

Parameter Descriptions

The following table describes the template and the function parameters.

Table 317: **densePyrOpticalFlow Function Parameter Descriptions**

Parameter	Description
NUM_PYR_LEVELS	Number of Image Pyramid levels used for the optical flow computation
NUM_LINES	Number of lines to buffer for the remap algorithm – used to find the temporal gradient
WINSIZE	Window Size over which Optical Flow is computed

Table 317: **densePyrOpticalFlow Function Parameter Descriptions** *(cont'd)*

Parameter	Description
FLOW_WIDTH, FLOW_INT	Data width and number of integer bits to define the signed flow vector data type. Integer bit includes the signed bit. The default type is 16-bit signed word with 10 integer bits and 6 decimal bits.
TYPE	Pixel type of the input image. XF_8UC1 is only the supported value.
ROWS	Maximum height or number of rows to build the hardware for this kernel
COLS	Maximum width or number of columns to build the hardware for this kernel
NPC	Number of pixels the hardware kernel must process per clock cycle. Only XF_NPPC1, 1 pixel per cycle, is supported.
USE_URAM	Enable to map some storage structures to UltraRAM
_curr_img	First input image stream
_next_img	Second input image to which the optical flow is computed with respect to the first image
_streamFlowin	32-bit Packed U and V flow vectors input for optical flow. The bits from 31-16 represent the flow vector U while the bits from 15-0 represent the flow vector V.
_streamFlowout	32-bit Packed U and V flow vectors output after optical flow computation. The bits from 31-16 represent the flow vector U while the bits from 15-0 represent the flow vector V.
level	Image pyramid level at which the algorithm is currently computing the optical flow.
scale_up_flag	Flag to enable the scaling-up of the flow vectors. This flag is set at the host when switching from one image pyramid level to the other.
scale_in	Floating point scale up factor for the scaling-up the flow vectors. The value is (previous_rows-1)/(current_rows-1). This is not 1 when switching from one image pyramid level to the other.
init_flag	Flag to initialize flow vectors to 0 in the first iteration of the highest pyramid level. This flag must be set in the first iteration of the highest pyramid level (smallest image in the pyramid). The flag must be unset for all the other iterations.

Resource Utilization

The following table summarizes the resource utilization of densePyrOpticalFlow for 1 pixel per cycle implementation, with the optical flow computed for a window size of 11 over an image size of 1920x1080 pixels. The results are after implementation in Vivado HLS 2019.1 for the Xilinx xczu9eg-ffvb1156-2L-e FPGA at 300 MHz.

Table 318: **densePyrOpticalFlow Function Resource Utilization Summary**

Operating Mode	Operating Frequency (MHz)	Utilization Estimate			
		LUTs	FFs	DSPs	BRAMs
1 Pixel	300	32231	16596	52	215

Resource Utilization with UltraRAM Enable

The following table summarizes the resource utilization of densePyrOpticalFlow for 1 pixel per cycle implementation, with the optical flow computed for a window size of 11 over an image size of 3840X2160 pixels. The results are after implementation in SDx 2019.1 for the Xilinx xczu7ev-ffvc1156-2 FPGA at 300 MHz with UltraRAM enabled.

Table 319: **densePyrOpticalFlow Function Resource Utilization Summary**

Operating Mode	Operating Frequency (MHz)	Utilization Estimate				
		LUTs	FFs	DSPs	BRAMs	URAM
1 Pixel	300	31164	42320	81	34	23

Performance Estimate

The following table summarizes performance figures on hardware for the densePyrOpticalFlow function for 5 iterations over 5 pyramid levels scaled down by a factor of two at each level. This has been tested on the zcu102 evaluation board.

Table 320: **densePyrOpticalFlow Function Performance Estimate Summary**

Operating Mode	Operating Frequency (MHz)	Image Size	Latency Estimate
			Max (ms)
1 pixel	300	1920x1080	49.7
1 pixel	300	1280x720	22.9
1 pixel	300	1226x370	12.02

Dense Non-Pyramidal LK Optical Flow

Optical flow is the pattern of apparent motion of image objects between two consecutive frames, caused by the movement of object or camera. It is a 2D vector field, where each vector is a displacement vector showing the movement of points from first frame to second.

Optical Flow works on the following assumptions:

- Pixel intensities of an object do not have too many variations in consecutive frames
- Neighboring pixels have similar motion

Consider a pixel I(x, y, t) in first frame. (Note that a new dimension, time, is added here. When working with images only, there is no need of time). The pixel moves by distance (dx, dy) in the next frame taken after time dt. Thus, since those pixels are the same and the intensity does not change, the following is true:

$$I(x, y, t) = I(x + dx, y + dy, t + dt)$$

Taking the Taylor series approximation on the right-hand side, removing common terms, and dividing by dt gives the following equation:

$$f_x u + f_y v + f_t = 0$$

Where $f_x = \frac{\delta f}{\delta x}$, $f_y = \frac{\delta f}{\delta x}$, $u = \frac{dx}{dt}$ and $v = \frac{dy}{dt}$.

The above equation is called the Optical Flow equation, where, f_x and f_y are the image gradientsand f_t is the gradient along time. However, (u, v) is unknown. It is not possible to solve this equation with two unknown variables. Thus, several methods are provided to solve this problem. One method is Lucas-Kanade. Previously it was assumed that all neighboring pixels have similar motion. The Lucas-Kanade method takes a patch around the point, whose size can be defined through the 'WINDOW_SIZE' template parameter. Thus, all the points in that patch have the same motion. It is possible to find (f_x, f_y, f_t) for these points. Thus, the problem now becomes solving 'WINDOW_SIZE * WINDOW_SIZE' equations with two unknown variables,which is over-determined. A better solution is obtained with the "least square fit" method. Below is the final solution, which is a problem with two equations and two unknowns:

$$\begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} \sum f_{x_i}^2 & \sum f_{x_i} f_{y_i} \\ \sum f_{x_i} f_{y_i} & \sum f_{y_i}^2 \end{bmatrix}^{-1} \begin{bmatrix} -\sum f_{x_i} f_{t_i} \\ -\sum f_{y_i} f_{t_i} \end{bmatrix}$$

API Syntax

```
template<int TYPE, int ROWS, int COLS, int NPC, int WINDOW_SIZE,bool
USE_URAM=false>
void DenseNonPyrLKOpticalFlow (xf::Mat<TYPE, ROWS, COLS, NPC> & frame0,
xf::Mat<TYPE, ROWS, COLS, NPC> & frame1, xf::Mat<XF_32FC1, ROWS, COLS, NPC>
& flowx, xf::Mat<XF_32FC1, ROWS, COLS, NPC> & flowy)
```

Parameter Descriptions

The following table describes the template and the function parameters.

Table 321: **DenseNonPyrLKOpticalFlow Function Parameter Descriptions**

Parameter	Description
Type	pixel type. The current supported pixel value is XF_8UC1, unsigned 8 bit.
ROWS	Maximum number of rows of the input image that the hardware kernel must be built for.
COLS	Maximum number of columns of the input image that the hardware kernel must be built for.
NPC	Number of pixels to process per cycle. Supported values are XF_NPPC1 (=1) and XF_NPPC2(=2).
WINDOW_SIZE	Window size over which optical flow will be computed. This can be any odd positive integer.
USE_URAM	Enable to map storage structures to UltraRAM.

Table 321: **DenseNonPyrLKOpticalFlow Function Parameter Descriptions** *(cont'd)*

Parameter	Description
frame0	First input images.
frame1	Second input image. Optical flow is computed between frame0 and frame1.
flowx	Horizontal component of the flow vectors. The format of the flow vectors is XF_32FC1 or single precision.
flowy	Vertical component of the flow vectors. The format of the flow vectors is XF_32FC1 or single precision.

Resource Utilization

The following table summarizes the resource utilization of DenseNonPyrLKOpticalFlow for a 4K image, as generated in the Vivado HLS 2019.1 version tool for the Xilinx Xczu9eg-ffvb1156-1-i-es1 FPGA at 300 MHz.

Table 322: **DenseNonPyrLKOpticalFlow Function Resource Utilization Summary**

Operating Mode	Operating Frequency (MHz)	Utilization Estimate			
		BRAM_18K	DSP_48Es	FF	LUTs
1 pixel	300	178	42	11984	7730
2 pixel	300	258	82	22747	15126

The following table summarizes the resource utilization of DenseNonPyrLKOpticalFlow for a 4K image, as generated in the SDx version tool for the Xilinx Xczu7eg-ffvb1156-1 FPGA at 300 MHz with UltraRAM enabled.

Table 323: **DenseNonPyrLKOpticalFlow Function Resource Utilization Summary with UltraRAM Eanble**

Operating Mode	Operating Frequency (MHz)	Utilization Estimate				
		BRAM_18K	URAM	DSP_48Es	FF	LUTs
1 pixel	300	0	12	42	11803	7469
2 pixel	300	0	23	80	22124	13800

Performance Estimate

The following table summarizes performance estimates of the DenseNonPyrLKOpticalFlow function for a 4K image, generated using Vivado HLS 2019.1 version tool for the Xilinx Xczu9eg-ffvb1156-1-i-es1 FPGA.

Table 324: **DenseNonPyrLKOpticalFlow Function Performance Estimate Summary**

Operating Mode	Operating Frequency (MHz)	Latency Estimate
		Max (ms)
1 pixel	300	28.01
2 pixel	300	14.01

Kalman Filter

The classic Kalman Filter is proposed for linear system. The state-space description of a linear system assumed to be:

$$x_{k+1} = A_k x_k + B_k u_k + \Gamma_k \xi_k$$
$$y_k = H_k x_k + \eta_k$$

where x_k is the state vector at k^{th} time instant, constant (known) A_k is an nxn state transition matrix, constant (known) B_k is an nxm control input matrix, constant (known) Γ_k is an nxp system noise input matrix, constant (known) H_k is a qxn measurement matrix, constant (known) with $1 \leq m, p, q \leq n$, $\{u_k\}$ a (known) sequence of m vectors (called a deterministic input sequence), and $\{\xi_k\}$ and $\{\eta_k\}$ are respectively, (unknown) system and observation noise sequences, with known statistical information such as mean, variance, and covariance.

The Kalman filter assumes the following:

1. $\{\xi_k\}$ and $\{\eta_k\}$ are assumed to be sequences of zero-mean Gaussian (or normal) white noise. That is, $E(\eta_k) = 0,\ E(\xi_k \xi_l^T) = Q_k \delta_{kl}$ and $E(\eta_k \eta_l^T) = R_k \delta_{kl}$, where δ_{kl} is a Kronecker Delta function, and Q_k and R_k are positive definite matrices, E(u) is an expectation of random variable u.
2. $E(\xi_k \eta_l^T) = 0\ \forall\ k, l$
3. The initial state x_0 is also assumed to be independent of $\{\xi_k\}$ and $\{\eta_k\}$, that is $E(x_0 \eta_k^T) = E(x_0 \xi_l^T) = 0\ \forall\ k, l$.

The representation $\hat{x}_{k|j} = \hat{x}_{k,\,j}$

The Kalman filter algorithm can be summarized as shown in the below equations: means the estimate of x at time instant k using all the data measured till the time instant j.

Initialization

$$\begin{cases} P_{0,0} = Var(x_0) \\ \hat{x}_{0|0} = E(x_0) \end{cases}$$

Time Update / Predict

$$\begin{cases} \hat{x}_{k|k-1} = A_{k-1}\hat{x}_{k-1|k-1} + B_{k-1}u_{k-1} \\ P_{k,\,k-1} = A_{k-1}P_{k-1,\,k-1}A_{k-1}^{\;T} + \Gamma_{k-1}Q_{k-1}\Gamma_{k-1}^{\;T} \end{cases}$$

Measurement Update/Correction

$$\begin{cases} G_k = P_{k,\,k-1}H_k^{\;T}\left(R_k + H_kP_{k,\,k-1}H_k^{\;T}\right)^{-1} \\ \hat{x}_{k|k} = \hat{x}_{k|k-1} + G_k\left(v_k - D_ku_k - H_k\hat{x}_{k|k-1}\right) \\ P_{k,\,k} = (I - G_kH_k)P_{k,\,k-1} \end{cases}$$

Where P is an estimate error covariance nxn matrix, G_k is Kalman gain nxq matrix, and k=1, 2,..

Computation Strategy

The numerical accuracy of the Kalman filter covariance measurement update is a concern for implementation, since it differentiates two positive definite arrays. This is a potential problem if finite precision is used for computation. This design uses UDU factorization of P to address the numerical accuracy/stability problems.

$$P_{k,\,k} = (I - G_kH_k)P_{k,\,k-1} = P_{k,\,k-1} - P_{k,\,k-1}H_k^{\;T}\left(R_k + H_kP_{k,\,k-1}H_k^{\;T}\right)^{-1}P_{k,\,k-1}$$

During the initialization (before the first iteration), the user has to supply error covariance matrix P's U0_mat and D0_mat matrices and system noise co-variance matrix Q's Uq_mat and Dq_mat matrices. These U and D matrices can be obtained using Backward Cholesky decomposition.

Below, we illustrate the Backward Cholesky decomposition of P into a unit upper triangular matrix U and diagonal matrix D such that P=UDUT.

For the *n* th column of U and D:

$$D_{nn} = P_{nn}$$

$$U_{in} = \begin{cases} 1, \quad i = n \\ P_{in} / \quad D_{nn}, \quad i = n-1,\ n-2,\ \dots 1 \end{cases}$$

For remaining columns, that is j=n-1, n-2, ...1,:

$$D_{jj} = P_{jj} - \sum_{k=j+1}^{n} D_{kk} U_{jk}^2$$

$$Uij = \begin{cases} 0, & i > j \\ 1, & i = j \\ \dfrac{\left[P_{ij} - \sum_{k=j+1}^{n} D_{kk} U_{ik} U_{jk}\right]}{D_{jj}}, & i = j-1, j-2, \ldots .1 \end{cases}$$

Example for Kalman Filter

```
//Control Flag
    INIT_EN       = 1; TIMEUPDATE_EN = 2; MEASUPDATE_EN = 4;
    XOUT_EN_TU      = 8; UDOUT_EN_TU     = 16; XOUT_EN_MU      = 32;
    UDOUT_EN_MU     = 64; EKF_MEM_OPT     = 128;
    //Load A_mat,B_mat,Uq_mat,Dq_mat,H_mat,X0_mat,U0_mat,D0_mat,R_mat
    //Initialization
KalmanFilter(A_mat, B_mat, Uq_mat, Dq_mat,  H_mat, X0_mat, U0_mat, D0_mat,
R_mat, u_mat, y_mat, Xout_mat, Uout_mat, Dout_mat, INIT_EN);

for(int iteration=0; iteration< count; iteration++)
{
    //Load u_mat (control input)
    for(int index=0; index <C_CTRL; index ++)
        u_mat.write_float(index, control_input[index]);

//Time Update
KalmanFilter(A_mat, B_mat, Uq_mat, Dq_mat,  H_mat, X0_mat, U0_mat, D0_mat,
R_mat, u_mat, y_mat, Xout_mat, Uout_mat, Dout_mat, TIMEUPDATE_EN +
XOUT_EN_TU + UDOUT_EN_TU);

//Load y_mat (measurement vector)
    for(int index =0; index <M_MEAS; index ++)
        y_mat.write_float(index, control_input[index]);

//Measurement Update
KalmanFilter(A_mat, B_mat, Uq_mat, Dq_mat,  H_mat, X0_mat, U0_mat, D0_mat,
R_mat, u_mat, y_mat, Xout_mat, Uout_mat, Dout_mat, MEASUPDATE_EN +
XOUT_EN_MU + UDOUT_EN_MU);
}
```

API Syntax

```
template<int N_STATE, int M_MEAS, int C_CTRL, Int  MTU, int MMU, bool
USE_URAM=0, bool EKF_EN=0, int TYPE, int NPC >
void KalmanFilter (    xf::Mat<TYPE, N_STATE, N_STATE, NPC>     &A_mat,
#if KF_C!=0
xf::Mat<TYPE, N_STATE, C_CTRL, NPC>     &B_mat,
#endif
xf::Mat<TYPE, N_STATE, N_STATE, NPC>     &Uq_mat,
xf::Mat<TYPE, N_STATE, 1, NPC>          &Dq_mat,
xf::Mat<TYPE, M_MEAS, N_STATE, NPC>     &H_mat,
xf::Mat<TYPE, N_STATE, 1, NPC>          &X0_mat,
xf::Mat<TYPE, N_STATE, N_STATE, NPC>     &U0_mat,
xf::Mat<TYPE, N_STATE, 1, NPC>          &D0_mat,
xf::Mat<TYPE, M_MEAS, 1, NPC>          &R_mat,
#if KF_C!=0
```

```
xf::Mat<TYPE, C_CTRL, 1, NPC>          &u_mat,
#endif
xf::Mat<TYPE, M_MEAS, 1, NPC>          &y_mat,
xf::Mat<TYPE, N_STATE, 1, NPC>          &Xout_mat,
xf::Mat<TYPE, N_STATE, N_STATE, NPC>      &Uout_mat,
xf::Mat<TYPE, N_STATE, 1, NPC>          &Dout_mat,
unsigned char flag)
```

Parameter Descriptions

Table 325: **Kalman Filter Function Parameter Descriptions**

Parameter	Used (✓) or Unused (X)			Description
	Initialization	Time Update	Measurement Update	
N_STATE	✓	✓	✓	Number of state variable; possible options are 1 to 128
M_MEAS	✓	✓	✓	Number of measurement variable; possible options are 1 to 128; M_MEAS must be less than or equal to N_STATE. In case of Extended Kalman Filter(EKF), M_MEAS should be 1.
C_CTRL	✓	✓	✓	Number of control variable; possible options are 0 to 128; C_CTRL must be less than or equal to N_STATE. In case of EKF, C_CTRL should be 1.
MTU	✓	✓	✓	Number of multipliers used in time update; possible options are 1 to 128; MTU must be less than or equal to N_STATE
MMU	✓	✓	✓	Number of multipliers used in Measurement update; possible options are 1 to 128; MMU must be less than or equal to N_STATE
USE_URAM	✓	✓	✓	URAM enable; possible options are 0 and 1
EKF_EN	✓	✓	✓	Extended Kalman Filter Enable; possible options are 0 and 1
TYPE	✓	✓	✓	Type of input pixel. Currently, only XF_32FC1 is supported.
NPC	✓	✓	✓	Number of pixels to be processed per cycle; possible option is XF_NPPC1 (NOT relevant for this function)
A_mat	✓	X	X	Transition matrix A. In case of EKF, Jacobian Matrix F is mapped to A_mat.
B_mat	✓	X	X	Control matrix B. In case of KF, B_mat argument is not required when C_CTRL=0. And in case of EKF, Dummy matrix with size (N_STATE x 1) is mapped to B_mat.
Uq_mat	✓	X	X	U matrix for Process noise covariance matrix Q
Dq_mat	✓	X	X	D matrix for Process noise covariance matrix Q(only diagonal elements)
H_mat	✓	X	X	Measurement Matrix H. In case of EKF, Jacobian Matrix H is mapped to H_mat.
X0_mat	✓	X	X	Initial state matrix. . In case of EKF, state transition function f is mapped to X0_mat.

Table 325: **Kalman Filter Function Parameter Descriptions** *(cont'd)*

Parameter	Used (✓) or Unused (X)			Description
	Initialization	Time Update	Measurement Update	
U0_mat	✓	X	X	U matrix for initial error estimate covariance matrix P
D0_mat	✓	X	X	D matrix for initial error estimate covariance matrix P(only diagonal elements)
R_mat	✓	X	X	Measurement noise covariance matrix R(only diagonal elements). In case of EKF, input only one value of R since M_MEAS=1.
u_mat	X	✓	X	Control input vector. In case of KF, u_mat argument is not required when C_CTRL=0. And in case of EKF, observation function h is mapped to u_mat.
y_mat	X	X	✓	Measurement vector. In case of EKF, input only one measurement since M_MEAS=1.
Xout_mat	X	✓	✓	Output state matrix
Uout_mat	X	✓	✓	U matrix for output error estimate covariance matrix P
Dout_mat	X	✓	✓	D matrix for output error estimate covariance matrix P(only diagonal elements)
flag	✓	✓	✓	Control flag register
All U, D counterparts of all initialized matrices (Q and P) are obtained using U-D factorization				

Table 326: **Control Flag Registers**

Flag bit	Description
0	Initialization enable
1	Time update enable
2	Measurement update enable
3	X_{out} enable for time update
4	U_{out}/D_{out} enable for time update
5	X_{out} enables for measurement update
6	U_{out}/D_{out} enable for measurement update
7	Read optimization (Uq_mat, Dq_mat, U0_mat, D0_mat and R_mat) for Extended Kalman Filter

Resource Utilization

The following table summarizes the resource utilization of the kernel in different configurations, generated using SDx 2019.1 tool for the Xilinx Xczu9eg-ffvb1156-1 FPGA.

Table 327: **Kalman Filter Function Resource Utilization Summary**

Name	Resource Utilization		
	N_STATE=128; C_CTRL=128; M_MEAS=128; MTU=24; MMU=24	N_STATE=64; C_CTRL=64; M_MEAS=12;MTU=16;MMU=16	N_STATE=5; C_CTRL=1; M_MEAS=3;MTU=2;MMU=2
	300 MHz	300 MHz	300 MHz
BRAM_18K	387	142	24
DSP48E	896	548	87
FF	208084	128262	34887
LUT	113556	70942	18141

The following table shows the resource utilization of the kernel for a configuration with USE_URAM enable, generated using SDx 2019.1 for the Xilinx xczu7ev-ffvc1156-2-e FPGA.

Table 328: **Resource Utilization with UltraRAM Enabled**

Resource	Resource Utilization (N_STATE=64; C_CTRL=64; M_MEAS=12; MTU=4; MMU=4) (300 MHz) (ms)
BRAM_18K	30
DSP48E	284
FF	99210
LUT	53939
URAM	11

Performance Estimate

The following table shows the performance of kernel for different configurations, as generated using SDx 2019.1 tool for the Xilinx® Xczu9eg-ffvb1156-1, for one iteration. Latency estimate is calculated by taking average latency of 100 iteration.

Table 329: **Kalman Filter Function Performance Estimate Summary**

Operating Mode	Latency Estimate	
	Operating Frequency (MHz)	Latency (ms)
N_STATE=128; C_CTRL=128; M_MEAS=128; MTU=24; MMU=24	300	0.7
N_STATE=64; C_CTRL=64; M_MEAS=12; MTU=16; MMU=16	300	0.12
N_STATE=5; C_CTRL=1; M_MEAS=3; MTU=2; MMU=2	300	0.04

The following table shows the performance of kernel for a configuration with UltraRAM enable, as generated using SDx 2019.1 tool for the Xilinx xczu7ev-ffvc1156-2-e, for one iteration. Latency estimate is calculated by taking average latency of 100 iteration.

Table 330: **Performance Estimate with UltraRAM**

Operating Mode	Operating Frequency (MHz)	Latency (ms)
N_STATE=64; C_CTRL=64; M_MEAS=12;MTU=4;MMU=4	300	0.25

Extended Kalman Filter

The Kalman filter estimates the state vector in a linear model. If the model is nonlinear, then a linearization procedure is performed to obtain the filtering equations. The Kalman filter so obtained will be called the Extended Kalman filter. A state-space description of non-linear system can have a non-linear model of the form:

$$x_{k+1} = (x_k) + T_k(x_k)\xi_k$$
$$z_k = h_k(x_k) + \eta_k$$

Where f_k and h_k are valued functions with ranges in R^n and R^q, respectively. 1≤q≤n, and T_k a matrix-valued function with range in $R^n x R^q$ such that for each *k* the first order partial derivatives of f_k (x_k) and h_k (x_k)with respect to all the components of x_k are continuous. We consider zero-mean Gaussian white noise sequences $\{\xi_k\}$ and $\{\eta_k\}$ with ranges in R^p and R^q respectively, 1≤p, q≤n.

The real-time linearization process is carried out as shown in the following equations. In the lines of the linear model, the initial estimate and predicted position are chosen to be:

$$\hat{x}_0 = E\left(x_0\right),\ \hat{x}_{1|0} = f_0\left(\hat{x}_0\right)$$

Then, $\hat{x}_k = \hat{x}_{k|k}$, consecutively, for k=1,2,..., use the predicted positions.

$$\hat{x}_{k|k-1} = f_{k-1}\left(\hat{x}_{k-1}\right)$$

Note:

1. $f_k(x_k) = \begin{bmatrix} f_k^1(x_k) \\ \vdots \\ f_k^n(x_k) \end{bmatrix}$, where $x_k = \begin{bmatrix} x_k^1 \\ \vdots \\ x_k^n \end{bmatrix}$, k is a time index and superscript is row index and

$$\left[\frac{\partial f_k(x_k)}{\partial x_k}\right] = \begin{bmatrix} \frac{\partial f_k^1(x_k)}{\partial x_k^1} & \cdots & \frac{\partial f_k^1(x_k)}{\partial x_k^n} \\ \vdots & \cdots & \vdots \\ \frac{\partial f_k^n(x_k)}{\partial x_k^1} & \cdots & \frac{\partial f_k^n(x_k)}{\partial x_k^n} \end{bmatrix}$$

2. R^m is a space of column vectors $x = [x_1 \ldots .x_m]^T$

The equation for time update computations is as follows:

$$P_{k|k-1} = \left[\frac{\partial f_{k-1}(\hat{x}_{k-1})}{\partial x_{k-1}}\right] P_{k-1|k-1} \left[\frac{\partial f_{k-1}(\hat{x}_{k-1})}{\partial x_{k-1}}\right]^T + T_{k-1}(\hat{x}_{k-1}) Q_{k-1} T_{k-1}(\hat{x}_{k-1})^T$$

$$= F_{k-1} P_{k-1|k-1} F_{k-1}^T + T_{k-1}(\hat{x}_{k-1}) Q_{k-1} T_{k-1}(\hat{x}_{k-1})^T$$

The equation for measurement update computations is as follows:

$$G_k = P_{k|k-1}\left[\frac{\partial h_k(\hat{x}_{k|k-1})}{\partial x_k}\right]^T \left(R_k + \left[\frac{\partial h_k(\hat{x}_{k|k-1})}{\partial x_k}\right] P_{k|k-1} \left[\frac{\partial h_k(\hat{x}_{k|k-1})}{\partial x_k}\right]^T\right)^{-1}$$

$$\hat{x}_{k|k} = \hat{x}_{k|k-1} + G_k(v_k - h_k(\hat{x}_{k|k-1}))$$

$$P_{k|k} = \left(I - G_k\left[\frac{\partial h_k(\hat{x}_{k|k-1})}{\partial x_k}\right]\right) P_{k|k-1} = (I - G_k H_k) P_{k|k-1}$$

Example for Example Kalman Filter

```
//Load F/B_mat/Uq_mat/Dq_mat/X0_mat/U0_mat/D0_mat

for(int iteration=0; iteration< count; iteration++)
{
    if(iteration ==0)
        model_fx(X0_mat, fx);// update fx using X0_mat
    else
model_fx(Xout_mat, fx); // update fx using Xout_mat

        unsigned char initFlag;
        if(iteration ==0)
            initFlag = INIT_EN;
        else
            initFlag = EKF_MEM_OPT+INIT_EN;

        //Initialization
KalmanFilter (F, B_mat, Uq_mat, Dq_mat, H, fx, U0_mat, D0_mat, R_mat, hx,
y_mat, Xout_mat, Uout_mat, Dout_mat, initFlag);

        //Time Update
KalmanFilter (F, B_mat, Uq_mat, Dq_mat, H, fx, U0_mat, D0_mat, R_mat, hx,
y_mat, Xout_mat, Uout_mat, Dout_mat, TIMEUPDATE_EN + XOUT_EN_TU +
UDOUT_EN_TU);
        for(int index=0; index< M_MEAS; index++)
        {
if(iteration ==0)
// update hx/H using X0_mat for one measurement at a time
            model_hxH(X0_mat, hx, H, index);
        else
```

```
        //update hx/H using Xout_mat for one measurement at a time
model_hxH(Xout_mat, hx, H, index);

//Load R_mat
            R_mat.write_float(0,R_matrix[index][index]);

            //Load y_mat
            Y_mat.write_float(0,measurement_vector[index]);

//Measurement Update
KalmanFilter (F, B_mat, Uq_mat, Dq_mat, H, fx, U0_mat, D0_mat, R_mat, hx,
y_mat, Xout_mat, Uout_mat, Dout_mat, MEASUPDATE_EN + XOUT_EN_MU +
UDOUT_EN_MU);
        }
}
```

Mean and Standard Deviation

The `meanStdDev` function computes the mean and standard deviation of input image. The output Mean value is in fixed point Q8.8 format, and the Standard Deviation value is in Q8.8 format. Mean and standard deviation are calculated as follows:

$$\mu = \frac{\sum_{y=0}^{height} \sum_{x=0}^{width} src(x, y)}{(width*height)}$$

$$\sigma = \sqrt{\frac{\sum_{y=0}^{height} \sum_{x=0}^{width} (\mu - src(x, y))^2}{(width*height)}}$$

API Syntax

```
template<int SRC_T,int ROWS, int COLS,int NPC=1>
void meanStdDev(xf::Mat<SRC_T, ROWS, COLS, NPC> & _src,unsigned short*
_mean,unsigned short* _stddev)
```

Parameter Descriptions

The following table describes the template and the function parameters.

Table 331: **meanStdDev Function Parameter Descriptions**

Parameter	Description
SRC_T	Input and Output pixel type. Only 8-bit, unsigned, 1 and 3 channels are supported (XF_8UC1 and XF_8UC3)
ROWS	Number of rows in the image being processed.
COLS	Number of columns in the image being processed. Must be a multiple of 8, for 8-pixel operation.

Table 331: **meanStdDev Function Parameter Descriptions** *(cont'd)*

Parameter	Description
NPC	Number of pixels to be processed per cycle; possible options are XF_NPPC1 and XF_NPPC8 for 1 pixel and 8 pixel operations respectively.
_src	Input image
_mean	16-bit data pointer through which the computed mean of the image is returned.
_stddev	16-bit data pointer through which the computed standard deviation of the image is returned.

Resource Utilization

The following table summarizes the resource utilization of the meanStdDev function, generated using Vivado HLS 2019.1 tool for the Xczu9eg-ffvb1156-1-i-es1 FPGA, to process a grayscale HD (1080x1920) image.

Table 332: **meanStdDev Function Resource Utilization Summary**

Operating Mode	Operating Frequency (MHz)	Utilization Estimate				
		BRAM_18K	DSP_48Es	FF	LUT	CLB
1 pixel	300	0	6	896	461	121
8 pixel	150	0	13	1180	985	208

The following table summarizes the resource utilization of the meanStdDev function, generated using Vivado HLS 2019.1 tool for the Xczu9eg-ffvb1156-1-i-es1 FPGA, to process a 4K 3Channel image.

Table 333: **meanStdDev Function Resource Utilization Summary**

Operating Mode	Operating Frequency (MHz)	Utilization Estimate				
		BRAM_18K	DSP_48Es	FF	LUT	CLB
1 pixel	300	0	7	5075	3324	725

Performance Estimate

The following table summarizes the performance in different configurations, as generated using Vivado HLS 2019.1 tool for the Xczu9eg-ffvb1156-1-i-es1, to process a grayscale HD (1080x1920) image.

Table 334: **meanStdDev Function Performance Estimate Summary**

Operating Mode	Latency Estimate
	Max Latency
1 pixel operation (300 MHz)	6.9 ms
8 pixel operation (150 MHz)	1.69 ms

Max

The Max function calculates the per-element maximum of two corresponding images src1, src2 and stores the result in dst.

$$dst(x,y)=max(src1(x,y) ,src2(x,y))$$

API Syntax

```
template< int SRC_T , int ROWS, int COLS, int NPC=1>
void Max(xf::Mat<SRC_T, ROWS, COLS, NPC> & _src1, xf::Mat<SRC_T, ROWS,
COLS, NPC> & _src2, xf::Mat<SRC_T, ROWS, COLS, NPC> & _dst)
```

Parameter Descriptions

The following table describes the template and the function parameters.

Table 335: **Max Function Parameter Descriptions**

Parameter	Description
SRC_T	Input Pixel Type. 8-bit, unsigned, 1 channel is supported (XF_8UC1).
ROWS	Maximum height of input and output image.
COLS	Maximum width of input and output image. In case of N-pixel parallelism, width should be multiple of N.
NPC	Number of pixels to be processed per cycle; possible options are XF_NPPC1 and XF_NPPC8 for 1 pixel and 8 pixel operations respectively.
_src1	First input image
_src2	Second input image
_dst	Output image

Resource Utilization

The following table summarizes the resource utilization of the Max function in Resource optimized (8 pixel) mode and normal mode as generated using Vivado HLS 2019.1 version tool for the Xczu9eg-ffvb1156-1-i-es1 FPGA.

Table 336: **Max Function Resource Utilization Summary**

Name	Resource Utilization	
	1 pixel per clock operation	8 pixel per clock operation
	300 MHz	150 MHz
BRAM_18K	0	0
DSP48E	0	0
FF	103	153
LUT	44	102
CLB	21	38

Performance Estimate

The following table summarizes a performance estimate of the kernel in different configurations, generated using Vivado HLS 2019.1 tool for Xczu9eg-ffvb1156-1-i-es1 FPGA to process a grayscale HD (1080x1920) image.

Table 337: **Max Function Performance Estimate Summary**

Operating Mode	Latency Estimate	
	Operating Frequency (MHz)	Latency (ms)
1 pixel	300	6.9
8 pixel	150	1.7

MaxS

The MaxS function calculates the maximum elements between src and given scalar value scl and stores the result in dst.

$$dst(I)=maxS(\ src(I)\ ,scl\ )$$

API Syntax

```
template< int SRC_T , int ROWS, int COLS, int NPC=1>
void MaxS(xf::Mat<SRC_T, ROWS, COLS, NPC> & _src1, unsigned char
_scl[XF_CHANNELS(SRC_T,NPC)], xf::Mat<SRC_T, ROWS, COLS, NPC> & _dst)
```

Parameter Descriptions

The following table describes the template and the function parameters.

Table 338: **MaxS Function Parameter Descriptions**

Parameter	Description
SRC_T	Input Pixel Type. 8-bit, unsigned, 1 channel is supported (XF_8UC1).
ROWS	Maximum height of input and output image.
COLS	Maximum width of input and output image. In case of N-pixel parallelism, width should be multiple of N.
NPC	Number of pixels to be processed per cycle; possible options are XF_NPPC1 and XF_NPPC8 for 1 pixel and 8 pixel operations respectively.
_src1	First Input image
_scl	Input scalar value, the size should be number of channels
_dst	Output image

Resource Utilization

The following table summarizes the resource utilization of the MaxS function in Resource optimized (8 pixel) mode and normal mode as generated using Vivado HLS 2019.1 version tool for the Xczu9eg-ffvb1156-1-i-es1 FPGA.

Table 339: **MaxS Function Resource Utilization Summary**

Name	Resource Utilization	
	1 pixel per clock operation	8 pixel per clock operation
	300 MHz	150 MHz
BRAM_18K	0	0
DSP48E	0	0
FF	162	43
LUT	103	104
CLB	32	20

Performance Estimate

The following table summarizes a performance estimate of the kernel in different configurations, generated using Vivado HLS 2019.1 tool for Xczu9eg-ffvb1156-1-i-es1 FPGA to process a grayscale HD (1080x1920) image.

Table 340: **MaxS Function Performance Estimate Summary**

Operating Mode	Latency Estimate	
	Operating Frequency (MHz)	Latency (ms)
1 pixel	300	6.9
8 pixel	150	1.7

Median Blur Filter

The function medianBlur performs a median filter operation on the input image. The median filter acts as a non-linear digital filter which improves noise reduction. A filter size of N would output the median value of the NxN neighborhood pixel values, for each pixel.

API Syntax

```
template<int FILTER_SIZE, int BORDER_TYPE, int TYPE, int ROWS, int COLS, 
int NPC>
void medianBlur (xf::Mat<TYPE, ROWS, COLS, NPC> & _src, xf::Mat<TYPE, ROWS, 
COLS, NPC> & _dst)
```

Parameter Descriptions

The following table describes the template and the function parameters.

Table 341: **medianBlur Function Parameter Descriptions**

Parameter	Description
FILTER_SIZE	Window size of the hardware filter for which the hardware kernel will be built. This can be any odd positive integer greater than 1.
BORDER_TYPE	The way in which borders will be processed in the hardware kernel. Currently, only XF_BORDER_REPLICATE is supported.
TYPE	Input and Output pixel type. Only 8-bit, unsigned, 1 and 3 channels are supported (XF_8UC1 and XF_8UC3)
ROWS	Number of rows in the image being processed.
COLS	Number of columns in the image being processed. Must be a multiple of 8, for 8-pixel operation.
NPC	Number of pixels to be processed in parallel. Options are XF_NPPC1 (for 1 pixel processing per clock), XF_NPPC8 (for 8 pixel processing per clock
_src	Input image.
_dst	Output image.

Resource Utilization

The following table summarizes the resource utilization of the medianBlur function for XF_NPPC1 and XF_NPPC8 configurations, generated using Vivado HLS 2019.1 version tool for the Xczu9eg-ffvb1156-1-i-es1 FPGA.

Table 342: **medianBlur Function Resource Utilization Summary**

Operating Mode	FILTER_SIZE	Operating Frequency (MHz)	Utilization Estimate			
			LUTs	FFs	DSPs	BRAMs
1 pixel	3	300	1197	771	0	3
8 pixel	3	150	6559	1595	0	6
1 pixel	5	300	5860	1886	0	5

The following table summarizes the resource utilization of the medianBlur function for XF_NPPC1 with 3channel image as input, generated using Vivado HLS 2019.1 version tool for the Xczu9eg-ffvb1156-1-i-es1 FPGA.

Table 343: **medianBlur Function Resource Utilization Summary**

Operating Mode	FILTER_SIZE	Operating Frequency (MHz)	Utilization Estimate			
			LUTs	FFs	DSPs	BRAMs
1 pixel	3	300	2100	1971	0	9
1 pixel	5	300	13541	9720	0	15

Performance Estimate

The following table summarizes performance estimates of medianBlur function on Vivado HLS 2019.1 version tool for the Xilinx Xczu9eg-ffvb1156-1-i-es1 FPGA.

Table 344: **medianBlur Function Performance Estimate Summary**

Operating Mode	FILTER_SIZE	Operating Frequency (MHz)	Input Image Size	Latency Estimate
				Max (ms)
1 pixel	3	300	1920x1080	6.99
8 pixel	3	150	1920x1080	1.75
1 pixel	5	300	1920x1080	7.00

Min

The Min function calculates the per element minimum of two corresponding images src1, src2 and stores the result in dst.

$$dst(I)=min(src1(I) ,src2(I))$$

API Syntax

```
template< int SRC_T , int ROWS, int COLS, int NPC=1>
void Min(xf::Mat<SRC_T, ROWS, COLS, NPC> & _src1, xf::Mat<SRC_T, ROWS,
COLS, NPC> & _src2, xf::Mat<SRC_T, ROWS, COLS, NPC> & _dst)
```

Parameter Descriptions

The following table describes the template and the function parameters.

Table 345: **Min Function Parameter Descriptions**

Parameter	Description
SRC_T	Input pixel type. 8-bit, unsigned, 1 channel is supported (XF_8UC1).
ROWS	Maximum height of input and output image
COLS	Maximum width of input and output image. Must be multiple of 8, for 8-pixel operation.
NPC	Number of pixels to be processed per cycle, possible options are XF_NPPC1 and XF_NPPC8 for 1 pixel and 8 pixel operations respectively.
_src1	First input image
_src2	Second input image
_dst	Output image

Resource Utilization

The following table summarizes the resource utilization of the Min function in Resource optimized (8 pixel) mode and normal mode as generated using Vivado HLS 2019.1 version tool for the Xczu9eg-ffvb1156-1-i-es1 FPGA.

Table 346: **Min Function Resource Utilization Summary**

Name	Resource Utilization	
	1 pixel per clock operation	8 pixel per clock operation
	300 MHz	150 MHz
BRAM_18K	0	0
DSP48E	0	0
FF	103	153
LUT	44	102
CLB	23	34

Performance Estimate

The following table summarizes a performance estimate of the kernel in different configurations, generated using Vivado HLS 2019.1 tool for Xczu9eg-ffvb1156-1-i-es1 FPGA to process a grayscale HD (1080x1920) image.

Table 347: **Min Function Performance Estimate Summary**

Operating Mode	Latency Estimate	
	Operating Frequency (MHz)	Latency (ms)
1 pixel	300	6.9
8 pixel	150	1.7

MinS

The MinS function calculates the minimum elements between src and given scalar value scl and stores the result in dst.

$$dst(x,y)=minS(\ src(x,y)\ ,scl\ )$$

API Syntax

```
template< int SRC_T , int ROWS, int COLS, int NPC=1>
void MinS(xf::Mat<SRC_T, ROWS, COLS, NPC> & _src1, unsigned char
_scl[XF_CHANNELS(SRC_T,NPC)], xf::Mat<SRC_T, ROWS, COLS, NPC> & _dst)
```

Parameter Descriptions

The following table describes the template and the function parameters.

Table 348: **MinS Function Parameter Descriptions**

Parameter	Description
SRC_T	Input Pixel Type. 8-bit, unsigned, 1 channel is supported (XF_8UC1).
ROWS	Maximum height of input and output image.
COLS	Maximum width of input and output image. In case of N-pixel parallelism, width should be multiple of N
NPC	Number of pixels to be processed per cycle; possible options are XF_NPPC1 and XF_NPPC8 for 1 pixel and 8 pixel operations respectively.
_src1	First Input image
_scl	Input scalar value, the size should be the number of channels.
_dst	Output image

Resource Utilization

The following table summarizes the resource utilization of the MinS function in Resource optimized (8 pixel) mode and normal mode as generated using Vivado HLS 2019.1 version tool for the Xczu9eg-ffvb1156-1-i-es1 FPGA

Table 349: **MinS Function Resource Utilization Summary**

Name	Resource Utilization	
	1 pixel per clock operation	8 pixel per clock operation
	300 MHz	150 MHz
BRAM_18K	0	0
DSP48E	0	0
FF	104	159
LUT	43	103
CLB	23	36

Performance Estimate

The following table summarizes a performance estimate of the kernel in different configurations, generated using Vivado HLS 2019.1 tool for Xczu9eg-ffvb1156-1-i-es1 FPGA to process a grayscale HD (1080x1920) image.

Table 350: **MinS Function Performance Estimate Summary**

Operating Mode	Latency Estimate	
	Operating Frequency (MHz)	Latency (ms)
1 pixel	300	6.9
8 pixel	150	1.7

MinMax Location

The `minMaxLoc` function finds the minimum and maximum values in an image and location of those values.

$$minVal = \min_{\substack{0 \le x' \le width \\ 0 \le y' \le height}} src(x', y')$$

$$maxVal = \max_{\substack{0 \le x' \le width \\ 0 \le y' \le height}} src(x', y')$$

API Syntax

```
template<int SRC_T,int ROWS,int COLS,int NPC>
void minMaxLoc(xf::Mat<SRC_T, ROWS, COLS, NPC> & _src,int32_t *max_value,
int32_t *min_value,uint16_t *_minlocx, uint16_t *_minlocy, uint16_t
*_maxlocx, uint16_t *_maxlocy )
```

Parameter Descriptions

The following table describes the template and the function parameters.

Table 351: **minMaxLoc Function Parameter Descriptions**

Parameter	Description
SRC_T	Input pixel type. 8-bit, unsigned, 1 channel (XF_8UC1), 16-bit, unsigned, 1 channel (XF_16UC1), 16-bit, signed, 1 channel (XF_16SC1), 32-bit, signed, 1 channel (XF_32SC1) are supported.
ROWS	Number of rows in the image being processed.
COLS	Number of columns in the image being processed.
NPC	Number of pixels to be processed per cycle; possible options are XF_NPPC1 and XF_NPPC8 for 1 pixel and 8 pixel operations respectively.
_src	Input image
max_val	Maximum value in the image, of type int.
min_val	Minimum value in the image, of type int.
_minlocx	x-coordinate location of the first minimum value.
_minlocy	y-coordinate location of the first minimum value.
_maxlocx	x-coordinate location of the first maximum value.
_maxlocy	y-coordinate location of the first maximum value.

Resource Utilization

The following table summarizes the resource utilization of the minMaxLoc function, generated using Vivado HLS 2019.1 tool for the Xczu9eg-ffvb1156-1-i-es1 FPGA, to process a grayscale HD (1080x1920) image.

Table 352: **minMaxLoc Function Resource Utilization Summary**

Operating Mode	Operating Frequency (MHz)	Utilization Estimate				
		BRAM_18K	DSP_48Es	FF	LUT	CLB
1 pixel	300	0	3	451	398	86
8 pixel	150	0	3	1049	1025	220

Performance Estimate

The following table summarizes the performance in different configurations, as generated using Vivado HLS 2019.1 tool for the Xilinx Xczu9eg-ffvb1156-1-i-es1, to process a grayscale HD (1080x1920) image.

Table 353: **minMaxLoc Function Performance Estimate Summary**

Operating Mode	Latency Estimate
	Max Latency
1 pixel operation (300 MHz)	6.9 ms
8 pixel operation (150 MHz)	1.69 ms

Mean Shift Tracking

Mean shift tracking is one of the basic object tracking algorithms. Mean-shift tracking tries to find the area of a video frame that is locally most similar to a previously initialized model. The object to be tracked is represented by a histogram. In object tracking algorithms target representation is mainly rectangular or elliptical region. It contains target model and target candidate. Color histogram is used to characterize the object. Target model is generally represented by its probability density function (pdf). Weighted RGB histogram is used to give more importance to object pixels.

Mean-shift algorithm is an iterative technique for locating the maxima of a density function. For object tracking, the density function used is the weight image formed using color histograms of the object to be tracked and the frame to be tested. By using the weighted histogram we are taking spatial position into consideration unlike the normal histogram calculation. This function will take input image pointer, top left and bottom right coordinates of the rectangular object, frame number and tracking status as inputs and returns the centroid using recursive mean shift approach.

API Syntax

```
template <int MAXOBJ, int MAXITERS, int OBJ_ROWS, int OBJ_COLS, int SRC_T,
int ROWS, int COLS, int NPC>
void MeanShift(xf::Mat<SRC_T, ROWS, COLS, NPC> &_in_mat, uint16_t* x1,
uint16_t* y1, uint16_t* obj_height, uint16_t* obj_width, uint16_t* dx,
uint16_t* dy, uint16_t* status, uint8_t frame_status, uint8_t no_objects,
uint8_t no_iters );
```

Template Parameter Descriptions

The following table describes the template parameters.

Table 354: **MeanShift Template Parameters**

Parameter	Description
MAXOBJ	Maximum number of objects to be tracked
MAXITERS	Maximum iterations for convergence
OBJ_ROWS	Maximum Height of the object to be tracked
OBJ_COLS	Maximum width of the object to be tracked
SRC_T	Type of the input xf::Mat, must be XF_8UC4, 8-bit data with 4 channels
ROWS	Maximum height of the image
COLS	Maximum width of the image
NPC	Number of pixels to be processed per cycle; this function supports only XF_NPPC1 or 1 pixel per cycle operations.

Function Parameter Description

The following table describes the function parameters.

Table 355: **MeanShift Function Parameters**

Parameter	Description
_in_mat	Input xF Mat
x1	Top Left corner x-coordinate of all the objects
y1	Top Left corner y-coordinate of all the objects
obj_height	Height of all the objects
obj_width	Width of all the objects
dx	Centers x-coordinate of all the objects returned by the kernel function
dy	Centers y-coordinate of all the objects returned by the kernel function
status	Track the object only if the status of the object is true, that is if the object goes out of the frame, status is made zero
frame_status	Set as zero for the first frame and one for other frames
no_objects	Number of objects racked
no_iters	Number of iterations for convergence

Resource Utilization and Performance Estimate

The following table summarizes the resource utilization of the MeanShift function for normal (1 pixel) configuration as generated in Vivado HLS 2019.1 release tool for the part xczu9eg-ffvb1156-i-es1 at 300 MHz to process a RGB image of resolution,1920x1080, and for 10 objects of size of 250x250 and 4 iterations.

Table 356: **MeanShift Function Resource Utilization and Performance Estimate Summary**

Configuration	Max. Latency (ms)	BRAMs	DSPs	FFs	LUTs
1 pixel	19.28	76	14	13198	10064

Limitations

The maximum number of objects that can be tracked is 10.

Otsu Threshold

Otsu threshold is used to automatically perform clustering-based image thresholding or the reduction of a gray-level image to a binary image. The algorithm assumes that the image contains two classes of pixels following bi-modal histogram (foreground pixels and background pixels), it then calculates the optimum threshold separating the two classes.

Otsu method is used to find the threshold which can minimize the intra class variance which separates two classes defined by weighted sum of variances of two classes.

$$\sigma_w^2(t) = w_1 \sigma_1^2(t) + w_2 \sigma_2^2(t)$$

Where, w_1is the class probability computed from the histogram.

$$w_1 = \sum_{1}^{t} p(i) \qquad w_2 = \sum_{t+1}^{I} p(i)$$

Otsu shows that minimizing the intra-class variance is the same as maximizing inter-class variance

$$\sigma_b^2 = \sigma - \sigma_w^2$$

$$\sigma_b^2 = w_1 w_2 (\mu_b - \mu_f)^2$$

Where, $\mu_b = \left[\sum_{1}^{t} p(i)x(i)\right] \Big/ w_1$, $\mu_f = \left[\sum_{t+1}^{I} p(i)x(i)\right] \Big/ w_2$ is the class mean.

API Syntax

```
template<int SRC_T, int ROWS, int COLS,int NPC=1> void
OtsuThreshold(xf::Mat<SRC_T, ROWS, COLS, NPC> & _src_mat, uint8_t &_thresh)
```

Parameter Descriptions

The following table describes the template and the function parameters.

Table 357: **OtsuThreshold Function Parameter Descriptions**

Parameter	Description
SRC_T	Input pixel type. Only 8-bit, unsigned, 1 channel is supported (XF_8UC1)
ROWS	Maximum height of input and output image.
COLS	Maximum width of input and output image (must be a multiple of 8, for 8-pixel operation)
NPC	Number of pixels to be processed per cycle; possible options are XF_NPPC1 and XF_NPPC8 for 1 pixel and 8 pixel operations respectively.
_src_mat	Input image
_thresh	Output threshold value after the computation

Resource Utilization

The following table summarizes the resource utilization of the OtsuThreshold function, generated using Vivado HLS 2019.1 tool for the Xilinx Xczu9eg-ffvb1156-1-i-es1 FPGA, to process a grayscale HD (1080x1920) image.

Table 358: **OtsuThreshold Function Resource Utilization Summary**

Operating Mode	Operating Frequency (MHz)	Utilization Estimate				
		BRAM_18K	DSP_48Es	FF	LUT	CLB
1 pixel	300	8	49	2239	3353	653
8 pixel	150	22	49	1106	3615	704

Performance Estimate

The following table summarizes the performance in different configurations, as generated using Vivado HLS 2019.1 tool for the Xilinx Xczu9eg-ffvb1156-1-i-es1, to process a grayscale HD (1080x1920) image.

Table 359: **OtsuThreshold Function Performance Estimate Summary**

Operating Mode	Latency Estimate
	Max Latency (ms)
1 pixel operation (300 MHz)	6.92 ms
8 pixel operation (150 MHz)	1.76 ms

Paintmask

The Paintmask function replace the pixel intensity value with given color value when mask is not zero or the corresponding pixel from the input image.

API Syntax

```
template< int SRC_T,int MASK_T, int ROWS, int COLS,int NPC=1>
void paintmask(xf::Mat<SRC_T, ROWS, COLS, NPC> & _src_mat, xf::Mat<MASK_T,
ROWS, COLS, NPC> & in_mask, xf::Mat<SRC_T, ROWS, COLS, NPC> & _dst_mat,
unsigned char _color[XF_CHANNELS(SRC_T,NPC)])
```

Parameter Descriptions

The following table describes the template and the function parameters.

Table 360: **Paintmask Function Parameter Descriptions**

Parameter	Description
SRC_T	Input pixel type. 8-bit, unsigned, 1 channel is supported (XF_8UC1).
MASK_T	Mask value type. 8-bit, unsigned, 1 channel is supported (XF_8UC1).
ROWS	Maximum height of input and output image
COLS	Maximum width of input and output image. In case of N-pixel parallelism, width should be multiple of N.
NPC	Number of pixels to be processed per cycle; possible options are XF_NPPC1 and XF_NPPC8 for 1 pixel and 8 pixel operations respectively.
_src_mat	Input image
_in_mask	Input mask image
_dst_mat	Output image
_color	Color value to be filled when mask is not zero

Resource Utilization

The following table summarizes the resource utilization of the Paintmask Resource optimized (8 pixel) mode and normal mode as generated using Vivado HLS 2019.1 version tool for the Xczu9eg-ffvb1156-1-i-es1 FPGA.

Table 361: **Paintmask Function Resource Utilization Summary**

Name	Resource Utilization	
	1 pixel per clock operation	8 pixel per clock operation
	300 MHz	150 MHz
BRAM_18K	0	0
DSP48E	0	0
FF	95	163
LUT	57	121

Table 361: **Paintmask Function Resource Utilization Summary** *(cont'd)*

Name	Resource Utilization	
	1 pixel per clock operation	8 pixel per clock operation
	300 MHz	150 MHz
CLB	14	33

Performance Estimate

The following table summarizes a performance estimate of the kernel in different configurations, generated using Vivado HLS 2019.1 tool for Xczu9eg-ffvb1156-1-i-es1 FPGA to process a grayscale HD (1080x1920) image.

Table 362: **Painmask Function Performance Estimate Summary**

Operating Mode	Latency Estimate	
	Operating Frequency (MHz)	Latency (ms)
1 pixel	300	6.9
8 pixel	150	1.7

Pixel-Wise Addition

The `add` function performs the pixel-wise addition between two input images and returns the output image.

$I_{out}(x, y) = I_{in1}(x, y) + I_{in2}(x, y)$

Where:

- $I_{out}(x, y)$ is the intensity of the output image at (x, y) position
- $I_{in1}(x, y)$ is the intensity of the first input image at (x, y) position
- $I_{in2}(x, y)$ is the intensity of the second input image at (x, y) position.

XF_CONVERT_POLICY_TRUNCATE: Results are the least significant bits of the output operand, as if stored in two's complement binary format in the size of its bit-depth.

XF_CONVERT_POLICY_SATURATE: Results are saturated to the bit depth of the output operand.

API Syntax

```
template<int POLICY_TYPE, int SRC_T, int ROWS, int COLS, int NPC=1>
void add (
xf::Mat<int SRC_T, int ROWS, int COLS, int NPC> src1,
xf::Mat<int SRC_T, int ROWS, int COLS, int NPC> src2,
xf::Mat<int SRC_T, int ROWS, int COLS, int NPC> dst )
```

Parameter Descriptions

The following table describes the template and the function parameters.

Table 363: **add Function Parameter Descriptions**

Parameter	Description
POLICY_TYPE	Type of overflow handling. It can be either, XF_CONVERT_POLICY_SATURATE or XF_CONVERT_POLICY_TRUNCATE.
SRC_T	pixel type. Options are XF_8UC1,XF_8UC3,XF_16SC3 and_16SC1..
ROWS	Maximum height of input and output image.
COLS	Maximum width of input and output image (must be a multiple of 8, for 8-pixel operation)
NPC	Number of pixels to be processed per cycle; possible options are XF_NPPC1 and XF_NPPC8 for 1 pixel and 8 pixel operations respectively.
src1	Input image
src2	Input image
dst	Output image

Resource Utilization

The following table summarizes the resource utilization in different configurations, generated using Vivado HLS 2019.1 tool for the Xilinx Xczu9eg-ffvb1156-1-i-es1 FPGA, to process a grayscale HD (1080x1920) image.

Table 364: **add Function Resource Utilization Summary**

Operating Mode	Operating Frequency (MHz)	Utilization Estimate				
		BRAM_18K	DSP_48Es	FF	LUT	CLB
1 pixel	300	0	0	62	55	11
8 pixel	150	0	0	65	138	24

The following table summarizes the resource utilization in different configurations, generated using Vivado HLS 2019.1 tool for the Xilinx Xczu9eg-ffvb1156-1-i-es1 FPGA, to process 4K image with 3 channels.

Table 365: **add Function Resource Utilization Summary**

Operating Mode	Operating Frequency (MHz)	Utilization Estimate				
		BRAM_18K	DSP_48Es	FF	LUT	CLB
1 pixel	300	0	0	113	77	24

Performance Estimate

The following table summarizes the performance in different configurations, as generated using Vivado HLS 2019.1 tool for the Xilinx Xczu9eg-ffvb1156-1-i-es1, to process a grayscale HD (1080x1920) image.

Table 366: **add Function Performance Estimate Summary**

Operating Mode	Latency Estimate
	Max Latency (ms)
1 pixel operation (300 MHz)	6.9
8 pixel operation (150MHz)	1.7

Pixel-Wise Multiplication

The `multiply` function performs the pixel-wise multiplication between two input images and returns the output image.

$I_{out}(x, y) = I_{in1}(x, y) * I_{in2}(x, y) * scale\_val$

Where:

- $I_{out}(x, y)$ is the intensity of the output image at (x, y) position
- $I_{in1}(x, y)$ is the intensity of the first input image at (x, y) position
- $I_{in2}(x, y)$ is the intensity of the second input image at (x, y) position
- scale_val is the scale value.

XF_CONVERT_POLICY_TRUNCATE: Results are the least significant bits of the output operand, as if stored in two's complement binary format in the size of its bit-depth.

XF_CONVERT_POLICY_SATURATE: Results are saturated to the bit depth of the output operand.

API Syntax

```
template<int POLICY_TYPE, int SRC_T,int ROWS, int COLS, int NPC=1>
void multiply (
xf::Mat<int SRC_T, int ROWS, int COLS, int NPC> src1,
xf::Mat<int SRC_T, int ROWS, int COLS, int NPC> src2,
xf::Mat<int SRC_T int ROWS, int COLS, int NPC> dst,
float scale)
```

Parameter Descriptions

The following table describes the template and the function parameters.

Table 367: **multiply Function Parameter Descriptions**

Parameter	Description
POLICY_TYPE	Type of overflow handling. It can be either, XF_CONVERT_POLICY_SATURATE or XF_CONVERT_POLICY_TRUNCATE.
SRC_T	pixel type. Options are XF_8UC1,XF_8UC3,XF_16SC1 and XF_16SC3.
ROWS	Maximum height of input and output image.
COLS	Maximum width of input and output image (must be a multiple of 8, for 8-pixel operation)
NPC	Number of pixels to be processed per cycle; possible options are XF_NPPC1 and XF_NPPC8 for 1 pixel and 8 pixel operations respectively.
src1	Input image
src2	Input image
dst	Output image
scale_val	Weighing factor within the range of 0 and 1

Resource Utilization

The following table summarizes the resource utilization in different configurations, generated using Vivado HLS 2019.1 tool for the Xilinx Xczu9eg-ffvb1156-1-i-es1 FPGA, to process a grayscale HD (1080x1920) image.

Table 368: **multiply Function Resource Utilization Summary**

Operating Mode	Operating Frequency (MHz)	Utilization Estimate				
		BRAM_18K	DSP_48Es	FF	LUT	CLB
1 pixel	300	0	2	124	59	18
8 pixel	150	0	16	285	108	43

The following table summarizes the resource utilization in different configurations, generated using Vivado HLS 2019.1 tool for the Xilinx Xczu9eg-ffvb1156-1-i-es1 FPGA, to process a 4K image with 3 channels.

Table 369: **multiply Function Resource Utilization Summary**

Operating Mode	Operating Frequency (MHz)	Utilization Estimate				
		BRAM_18K	DSP_48Es	FF	LUT	CLB
1 pixel	300	0	9	312	211	62

Performance Estimate

The following table summarizes the performance in different configurations, as generated using Vivado HLS 2019.1 tool for the Xilinx Xczu9eg-ffvb1156-1-i-es1, to process a grayscale HD (1080x1920) image.

Table 370: **multiply Function Performance Estimate Summary**

Operating Mode	Latency Estimate
	Max Latency (ms)
1 pixel operation (300 MHz)	6.9
8 pixel operation (150 MHz)	1.6

Pixel-Wise Subtraction

The `subtract` function performs the pixel-wise subtraction between two input images and returns the output image.

$I_{out}(x, y) = I_{in1}(x, y) - I_{in2}(x, y)$

Where:

- $I_{out}(x, y)$ is the intensity of the output image at (x, y) position
- $I_{in1}(x, y)$ is the intensity of the first input image at (x, y) position
- $I_{in2}(x, y)$ is the intensity of the second input image at (x, y) position.

XF_CONVERT_POLICY_TRUNCATE: Results are the least significant bits of the output operand, as if stored in two's complement binary format in the size of its bit-depth.

XF_CONVERT_POLICY_SATURATE: Results are saturated to the bit depth of the output operand.

API Syntax

```
template<int POLICY_TYPE int SRC_T, int ROWS, int COLS, int NPC=1>
void subtract (
xf::Mat<int SRC_T, int ROWS, int COLS, int NPC> src1,
xf::Mat<int SRC_T, int ROWS, int COLS, int NPC> src2,
xf::Mat<int SRC_T, int ROWS, int COLS, int NPC> dst )
```

Parameter Descriptions

The following table describes the template and the function parameters.

Table 371: **subtract Function Parameter Descriptions**

Parameter	Description
POLICY_TYPE	Type of overflow handling. It can be either, XF_CONVERT_POLICY_SATURATE or XF_CONVERT_POLICY_TRUNCATE.
SRC_T	pixel type. Options are XF_8UC1,XF_8UC3,XF_16SC3 and_16SC1.
ROWS	Maximum height of input and output image.
COLS	Maximum width of input and output image (must be a multiple of 8, for 8-pixel operation)
NPC	Number of pixels to be processed per cycle; possible options are XF_NPPC1 and XF_NPPC8 for 1 pixel and 8 pixel operations respectively.
src1	Input image
src2	Input image
dst	Output image

Resource Utilization

The following table summarizes the resource utilization in different configurations, generated using Vivado HLS 2019.1 tool for the Xilinx Xczu9eg-ffvb1156-1-i-es1 FPGA, to process a grayscale HD (1080x1920) image.

Table 372: **subtract Function Resource Utilization Summary**

Operating Mode	Operating Frequency (MHz)	Utilization Estimate				
		BRAM_18K	DSP_48Es	FF	LUT	CLB
1 pixel	300	0	0	62	53	11
8 pixel	150	0	0	59	13	21

The following table summarizes the resource utilization in different configurations, generated using Vivado HLS 2019.1 tool for the Xilinx Xczu9eg-ffvb1156-1-i-es1 FPGA, to process a 4K image with 3 channels.

Table 373: **subtract Function Resource Utilization Summary**

Operating Mode	Operating Frequency (MHz)	Utilization Estimate				
		BRAM_18K	DSP_48Es	FF	LUT	CLB
1 pixel	300	0	0	110	64	28

Performance Estimate

The following table summarizes the performance in different configurations, as generated using Vivado HLS 2019.1 tool for the Xilinx Xczu9eg-ffvb1156-1-i-es1, to process a grayscale HD (1080x1920) image.

Table 374: **subtract Function Performance Estimate Summary**

Operating Mode	Latency Estimate
	Max Latency (ms)
1 pixel operation (300 MHz)	6.9
8 pixel operation (150 MHz)	1.7

Reduce

The Reduce function reduces the matrix to a vector by treating rows/cols as set of 1-D vectors and performing specified operation on vectors until a single row/col is obtained.

Reduction operation could be one of the following:

- REDUCE_SUM : The output is the sum of all of the matrix's rows/columns.
- REDUCE_AVG : The output is the mean vector of all of the matrix's rows/columns.
- REDUCE_MAX : The output is the maximum (column/row-wise) of all of the matrix's rows/columns.
- REDUCE_MIN : The output is the minimum (column/row-wise) of all of the matrix's rows/columns.

API Syntax

```
template< int REDUCE_OP, int SRC_T , int DST_T,  int ROWS, int COLS, int
ONE_D_HEIGHT, int ONE_D_WIDTH,int NPC=1> void reduce(xf::Mat<SRC_T, ROWS,
COLS, NPC> & _src_mat, xf::Mat<DST_T, ONE_D_HEIGHT, ONE_D_WIDTH, 1> &
_dst_mat, unsigned char dim)
```

Parameter Descriptions

The following table describes the template and the function parameters.

Table 375: **Reduce Function Parameter Descriptions**

Parameter	Description
REDUCE_OP	The flag specifies the type of reduction operation to be applied.
SRC_T	Input pixel type. 8-bit, unsigned, 1 channel is supported (XF_8UC1).
DST_T	Output pixel type. 8-bit, unsigned, 1 channel is supported (XF_8UC1).
ROWS	Maximum height of input and output image

Table 375: **Reduce Function Parameter Descriptions** *(cont'd)*

Parameter	Description
COLS	Maximum width of input and output image. In case of N-pixel parallelism, width should be multiple of N.
ONE_D_HEIGHT	Height of output 1-D vector or reduced matrix
ONE_D_WIDTH	Width of output 1-D vector or reduced matrix
NPC	Number of pixels to be processed per cycle; possible option is XF_NPPC1 (1 pixel per cycle).
_src_mat	Input image
_dst_mat	1-D vector
dim	Dimension index along which the matrix is reduced. 0 means that the matrix is reduced to a single row. 1 means that the matrix is reduced to a single column.

Resource Utilization

The following table summarizes the resource utilization of the Reduce function Normal mode(1 pixel) as generated using Vivado HLS 2019.1 version tool for the Xczu9eg-ffvb1156-1-i-es1 FPGA.

Table 376: **Reduce Function Resource Utilization Summary**

Name	Resource Utilization
	1 pixel per clock operation
	300 MHz
BRAM_18K	2
DSP48E	0
FF	288
LUT	172
CLB	54

Performance Estimate

The following table summarizes a performance estimate of the kernel in different configurations, generated using Vivado HLS 2019.1 tool for Xczu9eg-ffvb1156-1-i-es1 FPGA to process a grayscale HD (1080x1920) image.

Table 377: **Reduce Function Performance Estimate Summary**

Operating Mode	Latency Estimate	
	Operating Frequency (MHz)	Latency (ms)
1 pixel	300	6.9

Remap

The `remap` function takes pixels from one place in the image and relocates them to another position in another image. Two types of interpolation methods are used here for mapping the image from source to destination image.

$$dst = src\left(map_x(x,\ y),\ map_y(x,\ y)\right)$$

API Syntax

```
template<int WIN_ROWS,int INTERPOLATION_TYPE, int SRC_T, int MAP_T, int
DST_T, int ROWS, int COLS, int NPC = 1,bool USE_URAM=false>

void remap (xf::Mat<SRC_T, ROWS, COLS, NPC> &_src_mat,
          xf::Mat<DST_T, ROWS, COLS, NPC> &_remapped_mat,
          xf::Mat<MAP_T, ROWS, COLS, NPC> &_mapx_mat,
          xf::Mat<MAP_T, ROWS, COLS, NPC> &_mapy_mat);
```

Parameter Descriptions

The following table describes the template parameters.

Table 378: **remap template Parameter Descriptions**

Parameter	Description
WIN_ROWS	Number of input image rows to be buffered inside. Must be set based on the map data. For instance, for left right flip, 2 rows are sufficient.
INTERPOLATION_TYPE	Type of interpolation, either XF_INTERPOLATION_NN (nearest neighbor) or XF_INTERPOLATION_BILINEAR (linear interpolation)
SRC_T	Input and Output pixel type. Only 8-bit, unsigned, 1 and 3 channels are supported (XF_8UC1 and XF_8UC3)
MAP_T	Map type. Single channel float type. XF_32FC1.
DST_T	Output image type. Grayscale image of type 8-bits and single channel. XF_8UC1.
ROWS	Height of input and output images
COLS	Width of input and output images
NPC	Number of pixels to be processed per cycle; this function supports only XF_NPPC1 or 1 pixel per cycle operations.
USE_URAM	Enable to map some structures to UltraRAM instead of BRAM.

The following table describes the function parameters.

Table 379: **remap Function Parameter Descriptions**

PARAMETERS	DESCRIPTION
_src_mat	Input xF Mat
_remapped_mat	Output xF Mat
_mapx_mat	mapX Mat of float type

Table 379: **remap Function Parameter Descriptions** *(cont'd)*

PARAMETERS	DESCRIPTION
_mapy_mat	mapY Mat of float type

Resource Utilization

The following table summarizes the resource utilization of remap, for HD (1080x1920) images generated in the Vivado HLS 2019.1 version tool for the Xilinx xczu9eg-ffvb1156-i-es1 FPGA at 300 MHz, with WIN_ROWS as 64 for the XF_INTERPOLATION_BILINEAR mode.

Table 380: **remap Function Resource Utilization Summary**

Name	Resource Utilization
BRAM_18K	64
DSP48E	17
FF	1738
LUT	1593
CLB	360

The following table summarizes the resource utilization of remap, for 4K (3840x2160) images generated in the SDx 2019.1 version tool for the Xilinx xczu7ev-ffvc1156 FPGA at 300 MHz, with WIN_ROWS as 100 for the XF_INTERPOLATION_BILINEAR mode using UltraRAM .

Table 381: **remap Function Resource Utilization Summary with UltraRAM Enabled**

Name	Resource Utilization
BRAM_18K	3
DSP48E	10
URAM	24
FF	3196
LUT	3705

Performance Estimate

The following table summarizes the performance of remap(), for HD (1080x1920) images generated in the Vivado HLS 2019.1 version tool for the Xilinx xczu9eg-ffvb1156-i-es1 FPGA at 300 MHz, with WIN_ROWS as 64 for XF_INTERPOLATION_BILINEAR mode.

Table 382: **remap Function Performance Estimate Summary**

Operating Mode	Operating Frequency (MHz)	Latency Estimate Max latency (ms)
1 pixel mode	300	7.2

Resolution Conversion (Resize)

Resolution Conversion is the method used to resize the source image to the size of the destination image. Different types of interpolation techniques can be used in resize function, namely: Nearest-neighbor, Bilinear, and Area interpolation. The type of interpolation can be passed as a template parameter to the API. The following enumeration types can be used to specify the interpolation type:

- XF_INTERPOLATION_NN - For Nearest-neighbor interpolation
- XF_INTERPOLATION_BILINEAR - For Bilinear interpolation
- XF_INTERPOLATION_AREA - For Area interpolation

Note: Scaling factors greater than or equal to 0.25 are supported in down-scaling and values less than or equal to 8 are supported for up-scaling.

API Syntax

```
template<int INTERPOLATION_TYPE, int TYPE, int SRC_ROWS, int SRC_COLS, int
DST_ROWS, int DST_COLS, int NPC,int MAX_DOWN_SCALE>
void resize (xf::Mat<TYPE, SRC_ROWS, SRC_COLS, NPC> & _src, xf::Mat<TYPE,
DST_ROWS, DST_COLS, NPC> & _dst)
```

Parameter Descriptions

The following table describes the template and the function parameters.

Table 383: **resize Function Parameter Descriptions**

Parameter	Description
INTERPOLATION_TYPE	Interpolation type. The different options possible are • XF_INTERPOLATION_NN – Nearest Neighbor Interpolation • XF_INTERPOLATION_BILINEAR – Bilinear interpolation • XF_INTERPOLATION_AREA – Area Interpolation
TYPE	Input and Output pixel type. Only 8-bit, unsigned, 1 and 3 channels are supported (XF_8UC1 and XF_8UC3)
SRC_ROWS	Maximum Height of input image for which the hardware kernel would be built.

Table 383: **resize Function Parameter Descriptions** *(cont'd)*

Parameter	Description
SRC_COLS	Maximum Width of input image for which the hardware kernel would be built (must be a multiple of 8).
DST_ROWS	Maximum Height of output image for which the hardware kernel would be built.
DST_COLS	Maximum Width of output image for which the hardware kernel would be built (must be a multiple of 8).
NPC	Number of pixels to be processed per cycle. Possible options are XF_NPPC1 (1 pixel per cycle) and XF_NPPC8 (8 pixel per cycle).
MAX_DOWN_SCALE	Set to 2 for all 1 pixel modes, and for upscale in x direction. When down scaling in x direction in 8-pixel mode, please set this parameter to the next highest integer value of the down scale factor i.e., if downscaling from 1920 columns to 1280 columns, set to 2. For 1920 to 640, set to 3.
_src	Input Image
_dst	Output Image

Resource Utilization

The following table summarizes the resource utilization of Resize function in Resource Optimized (8 pixel) mode and Normal mode, as generated in the Vivado HLS 2019.1 tool for the Xilinx xczu9eg-ffvb1156-2-i-es2 FPGA.

Table 384: **resize Function Resource Utilization Summary**

Operating Mode	Utilization Estimate									
	1 Pixel (at 300 MHz)					8 Pixel (at 150MHz)				
	IMAGESIZE	LUTs	FFs	DSPs	BRAMs	IMAGESIZE	LUTs	FFs	DSPs	BRAMs
Downscale Nearest Neighbor	1920X1080 TO 960X1620	1089	1593	4	2	3840X2160 TO 1920X1080	2545	2250	4	12
Downscale Bilinear	1920X1080 TO 960X1080	1340	1846	8	2	3840X2160 TO 1920X1080	5159	3092	36	12
Downscale Area	3840X2160 TO 1920X1080	2341	3550	44	24	Configuration not supported				
Upscale Nearest Neighbor	1920X1080 TO 3840X540	1089	1593	4	2	1920X1080 TO 3840X2160	1818	1686	4	6
Upscale Bilinear	1920X1080 TO 3840X540	1340	1846	8	2	1920X1080 TO 3840X2160	3697	2739	36	6
Upscale Area	1920X1080 TO 3840X2160	1312	2220	16	12	Configuration not supported				

The following table summarizes the resource utilization of Resize function in Normal mode, as generated in the Vivado HLS 2019.1 tool for the Xilinx xczu9eg-ffvb1156-2-i-es2 FPGA for 3channel image as input.

Table 385: **resize Function Resource Utilization Summary**

Operating Mode	Utilization Estimate 1 Pixel (at 300 MHz)				
	IMAGESIZE	LUTs	FFs	DSPs	BRAMs
Downscale Nearest Neighbor	3840X2160 TO 1920X108	1184	168	4	18
Downscale Bilinear	3840X2160 TO 1920X1080	1592	2058	14	18
Downscale Area	3840X2160 TO 1920X1080	3212	4777	104	72
Upscale Nearest Neighbor	1920X1080 TO 3840X2160	1166	1697	4	9
Upscale Bilinear	1920X1080 TO 3840X2160	1574	2053	14	9
Upscale Area	1920X1080 TO 3840X2160	1731	2733	36	31

Performance Estimate

The following table summarizes the performance estimation of Resize for various configurations, as generated in the Vivado HLS 2019.1 tool for the xczu9eg-ffvb1156-2-i-es2 FPGA at 300 MHz to resize a grayscale image from 1080x1920 to 480x640 (downscale); and to resize a grayscale image from 1080x1920 to 2160x3840 (upscale). This table also shows the latencies obtained for different interpolation types.

Table 386: **resize Function Performance Estimate Summary**

Operating Mode	Operating Frequency (MHz)	Latency Estimate (ms)					
		Downscale NN	Downscale Bilinear	Downscale Area	Upscale NN	Upscale Bilinear	Upscale Area
1 pixel	300	6.94	6.97	7.09	27.71	27.75	27.74

BGR2HSV

The `BGR2HSV` function converts the input image color space to HSV color space and returns the HSV image as the output.

API Syntax

```
template<int SRC_T, int ROWS, int COLS,int NPC=1>
          void BGR2HSV(xf::Mat<SRC_T, ROWS, COLS, NPC> &
_src_mat,xf::Mat<SRC_T, ROWS, COLS, NPC> & _dst_mat)
```

Parameter Descriptions

The table below describes the template and the function parameters.

Parameter	Description
SRC_T	Input pixel type should be XF_8UC3
DST_T	Output pixel type should be XF_8UC3
ROWS	Maximum height of input and output image
COLS	Maximum width of input and output image. Must be multiple of 8, for 8-pixel operation.
NPC	Number of pixels to be processed per cycle. Only XF_NPPC1 is supported.
_src_mat	Input image
_dst_mat	Output image

convertScaleAbs

The `convertScaleAbs` function converts an input image src with optional linear transformation, save the result as image dst.

$$dst(x,y)= src1(x,y)*scale+shift$$

API Syntax

```
template< int SRC_T,int DST_T, int ROWS, int COLS, int NPC = 1>
void convertScaleAbs(xf::Mat<SRC_T, ROWS, COLS, NPC> & src1, xf::Mat<DST_T,
ROWS, COLS, NPC> & dst,float scale, float shift)
```

Parameter Descriptions

The following table describes the template and the function parameters.

Table 387: **convertScaleAbs Function Parameter Descriptions**

Parameter	Description
SRC_T	Input pixel type. Only 8-bit, unsigned, 1 channel is supported (XF_8UC1).
DST_T	Output pixel type. Only 8-bit, unsigned, 1 channel is supported (XF_8UC1).
ROWS	Maximum height of input and output image.
COLS	Maximum width of input and output image. In case of N-pixel parallelism, width should be multiple of N.
NPC	Number of pixels to be processed per cycle; possible options are XF_NPPC1 and XF_NPPC8 for 1 pixel and 8 pixel operations respectively.
src1	Input image
scale	Scale factor
shift	Delta/shift added to scaled value.
dst	Output image

Resource Utilization

The following table summarizes the resource utilization of the convertScaleAbs function in Resource optimized (8 pixel) mode and normal mode as generated using Vivado HLS 2019.1 version tool for the Xczu9eg-ffvb1156-1-i-es1 FPGA.

Table 388: **convertScaleAbs Function Resource Utilization Summary**

Name	Resource Utilization	
	1 pixel per clock operation	8 pixel per clock operation
	300 MHz	150 MHz
BRAM_18K	0	0
DSP48E	10	38
FF	949	1971
LUT	1052	1522
CLB	218	382

Performance Estimate

The following table summarizes a performance estimate of the kernel in different configurations, generated using Vivado HLS 2019.1 tool for Xczu9eg-ffvb1156-1-i-es1 FPGA to process a grayscale HD (1080x1920) image...

Table 389: **convertScaleAbs Function Performance Estimate Summary**

Operating Mode	Latency Estimate	
	Operating Frequency (MHz)	Latency (ms)
1 pixel	300	6.9
8 pixel	150	1.7

Scharr Filter

The `Scharr` function computes the gradients of input image in both x and y direction by convolving the kernel with input image being processed.

For Kernel size 3x3:

- GradientX:

$$G_x = \begin{bmatrix} -3 & 0 & 3 \\ -10 & 0 & 10 \\ -3 & 0 & 3 \end{bmatrix} * I$$

- GradientY:

$$G_y = \begin{bmatrix} -3 & -10 & -3 \\ 0 & 0 & 0 \\ 3 & 10 & 3 \end{bmatrix} * I$$

API Syntax

```
template<int BORDER_TYPE, int SRC_T,int DST_T, int ROWS, int COLS,int NPC=1>
void Scharr(xf::Mat<SRC_T, ROWS, COLS, NPC> & _src_mat,xf::Mat<DST_T, ROWS,
COLS, NPC> & _dst_matx,xf::Mat<DST_T, ROWS, COLS, NPC> & _dst_maty)
```

Parameter Descriptions

The following table describes the template and the function parameters.

Table 390: **Scharr Function Parameter Descriptions**

Parameter	Description
BORDER_TYPE	Border type supported is XF_BORDER_CONSTANT
SRC_T	Input pixel type. Only 8-bit, unsigned, 1 and 3 channels are supported (XF_8UC1 and XF_8UC3)
DST_T	Output pixel type. Only 8-bit unsigned, 16-bit signed,1 and 3 channels are supported (XF_8UC1, XF_16SC1,XF_8UC3 and XF_16SC3)
ROWS	Maximum height of input and output image
COLS	Maximum width of input and output image. Must be multiple of 8, for 8-pixel operation.
NPC	Number of pixels to be processed per cycle; possible options are XF_NPPC1 and XF_NPPC8 for 1 pixel and 8 pixel operations respectively.
_src_mat	Input image
_dst_matx	X gradient output image.
_dst_maty	Y gradient output image.

Resource Utilization

The following table summarizes the resource utilization of the kernel in different configurations, generated using Vivado HLS 2019.1 tool for the Xilinx Xczu9eg-ffvb1156-1-i-es1 FPGA, to process a grayscale HD (1080x1920) image.

Table 391: **Scharr Function Resource Utilization Summary**

Name	Resource Utilization	
	1 pixel	8 pixel
	300 MHz	150 MHz
BRAM_18K	3	6
DSP48E	0	0
FF	728	1434
LUT	812	2481
CLB	171	461

The following table summarizes the resource utilization of the kernel in different configurations, generated using Vivado HLS 2019.1 tool for the Xilinx Xczu9eg-ffvb1156-1-i-es1 FPGA, to process a 4K 3 channel image.

Table 392: **Scharr Function Resource Utilization Summary**

Name	Resource Utilization
	1 pixel
	300 MHz
BRAM_18K	18
DSP48E	0
FF	1911
LUT	1392

Performance Estimate

The following table summarizes the performance of the kernel in different configurations, as generated using Vivado HLS 2019.1 tool for the Xilinx Xczu9eg-ffvb1156-1-i-es1, to process a grayscale HD (1080x1920) image.

Table 393: **Scharr Function Performance Estimate Summary**

Operating Mode	Operating Frequency (MHz)	Latency (ms)
1 pixel	300	7.2
8 pixel	150	1.7

Set

The Set function sets the each pixel in input image to a given scalar value and stores the result in dst.

API Syntax

```
template< int SRC_T , int ROWS, int COLS, int NPC=1>
void set(xf::Mat<SRC_T, ROWS, COLS, NPC> & _src1, unsigned char
_scl[XF_CHANNELS(SRC_T,NPC)], xf::Mat<SRC_T, ROWS, COLS, NPC> & _dst)
```

Parameter Descriptions

The following table describes the template and the function parameters.

Table 394: **Set Function Parameter Descriptions**

Parameter	Description
SRC_T	Input pixel type. 8-bit, unsigned, 1 channel is supported (XF_8UC1).
ROWS	Maximum height of input and output image
COLS	Maximum width of input and output image. Must be multiple of 8, for 8-pixel operation.
NPC	Number of pixels to be processed per cycle.
_src1	First input image
_scl	Scalar value
_dst	Output image

Resource Utilization

The following table summarizes the resource utilization of the Set function in Resource optimized (8 pixel) mode and normal mode as generated using Vivado HLS 2019.1 version tool for the Xczu9eg-ffvb1156-1-i-es1 FPGA.

Table 395: **Set Function Resource Utilization Summary**

Name	Resource Utilization	
	1 pixel per clock operation	8 pixel per clock operation
	300 MHz	150 MHz
BRAM_18K	0	0
DSP48E	0	0
FF	87	87
LUT	43	42
CLB	17	18

Performance Estimate

The following table summarizes a performance estimate of the kernel in different configurations, generated using Vivado HLS 2019.1 tool for Xczu9eg-ffvb1156-1-i-es1 FPGA to process a grayscale HD (1080x1920) image.

Table 396: **Set Function Performance Estimate Summary**

Operating Mode	Latency Estimate	
	Operating Frequency (MHz)	Latency (ms)
1 pixel	300	6.9
8 pixel	150	1.7

Sobel Filter

The `Sobel` function Computes the gradients of input image in both x and y direction by convolving the kernel with input image being processed.

- For Kernel size 3x3
  - GradientX:

$$G_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} * I$$

  - GradientY:

$$G_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix} * I$$

- For Kernel size 5x5
  - GradientX:

$$G_x = \begin{bmatrix} -1 & -2 & 0 & 2 & 1 \\ -4 & -8 & 0 & 8 & 4 \\ -6 & -12 & 0 & 12 & 6 \\ -4 & -8 & 0 & 8 & 4 \\ -1 & -2 & 0 & 2 & 1 \end{bmatrix} * I$$

  - GradientY:

$$G_y = \begin{bmatrix} -1 & -4 & -6 & -4 & -1 \\ -2 & -8 & -12 & -8 & -2 \\ 0 & 0 & 0 & 0 & 0 \\ 2 & 8 & 12 & 8 & 2 \\ 1 & 4 & 6 & 4 & 1 \end{bmatrix} * I$$

- For Kernel size 7x7
  - GradientX:

$$G_x = \begin{bmatrix} -1 & -4 & -5 & 0 & 5 & 4 & 1 \\ -6 & -24 & -30 & 0 & 30 & 24 & 6 \\ -15 & -60 & 75 & 0 & 75 & 60 & 15 \\ -20 & -80 & -100 & 0 & 75 & 60 & 15 \\ -15 & -60 & -75 & 0 & 75 & 60 & 15 \\ -6 & -24 & -30 & 0 & 30 & 24 & 6 \\ -1 & -4 & -5 & 0 & 5 & 4 & 1 \end{bmatrix} * I$$

- GradientY:

$$G_y = \begin{bmatrix} -1 & -6 & -15 & -20 & -15 & -6 & -1 \\ -4 & -24 & -60 & -80 & -60 & -24 & -4 \\ -5 & -30 & -75 & -100 & -75 & -30 & -5 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 5 & 30 & 75 & 100 & 75 & 30 & 5 \\ 4 & 24 & 60 & 80 & 60 & 24 & 4 \\ 1 & 6 & 15 & 20 & 15 & 6 & 1 \end{bmatrix} * I$$

API Syntax

```
template<int BORDER_TYPE,int FILTER_TYPE, int SRC_T,int DST_T, int ROWS,
int COLS,int NPC=1,bool USE_URAM=false>
void Sobel(xf::Mat<SRC_T, ROWS, COLS, NPC> & _src_mat,xf::Mat<DST_T, ROWS,
COLS, NPC> & _dst_matx,xf::Mat<DST_T, ROWS, COLS, NPC> & _dst_maty)
```

Parameter Descriptions

The following table describes the template and the function parameters.

Table 397: **Sobel Function Parameter Descriptions**

Parameter	Description
FILTER_TYPE	Filter size. Filter size of 3 (XF_FILTER_3X3), 5 (XF_FILTER_5X5) and 7 (XF_FILTER_7X7) are supported.
BORDER_TYPE	Border Type supported is XF_BORDER_CONSTANT
SRC_T	Input pixel type. Only 8-bit, unsigned, 1 and 3 channels are supported (XF_8UC1 and XF_8UC3)
DST_T	Output pixel type. Only 8-bit unsigned, 16-bit signed,1 and 3 channels are supported (XF_8UC1, XF_16SC1,XF_8UC3 and XF_16SC3)
ROWS	Maximum height of input and output image.
COLS	Maximum width of input and output image. Must be multiple of 8, for 8-pixel operation.
NPC	Number of pixels to be processed per cycle; possible options are XF_NPPC1 and XF_NPPC8 for 1 pixel and 8 pixel operations respectively.
USE_URAM	Enable to map storage structures to UltraRAM
_src_mat	Input image

Table 397: **Sobel Function Parameter Descriptions** *(cont'd)*

Parameter	Description
_dst_matx	X gradient output image.
_dst_maty	Y gradient output image.

1. Sobel 7x7 8-pixel is not supported.

Resource Utilization

The following table summarizes the resource utilization of the kernel in different configurations, generated using Vivado HLS 2019.1 tool for the Xilinx Xczu9eg-ffvb1156-1-i-es1 FPGA, to process a grayscale HD (1080x1920) image.

Table 398: **Sobel Function Resource Utilization Summary**

Operating Mode	Filter Size	Operating Frequency (MHz)	Utilization Estimate				
			BRAM_18K	DSP_48Es	FF	LUT	CLB
1 pixel	3x3	300	3	0	609	616	135
	5x5	300	5	0	1133	1499	308
	7x7	300	7	0	2658	3334	632
8 pixel	3x3	150	6	0	1159	1892	341
	5x5	150	10	0	3024	5801	999

The following table summarizes the resource utilization of the kernel in different configurations, generated using Vivado HLS 2019.1 tool for the Xilinx Xczu9eg-ffvb1156-1-i-es1 FPGA, to process a 4K 3 Channel image.

Table 399: **Sobel Function Resource Utilization Summary**

Operating Mode	Filter Size	Operating Frequency (MHz)	Utilization Estimate			
			BRAM_18K	DSP_48Es	FF	LUT
1 pixel	3x3	300	18	0	1047	1107
	5x5	300	30	0	5370	3312
	7x7	300	42	0	6100	5496

The following table summarizes the resource utilization of the kernel in different configurations, generated using SDx 2019.1 tool for the Xilinx xczu7ev-ffvc1156-2-e FPGA, to process a grayscale 4K (3840x2160) image with UltraRAM enable.

Table 400: **Sobel Function Resource Utilization Summary with UltraRAM enable**

Operating Mode	Filter Size	Operating Frequency (MHz)	Utilization Estimate				
			BRAM_18K	URAM	DSP_48Es	FF	LUT
1 pixel	3x3	300	0	1	0	919	707
	5x5	300	0	1	0	2440	1557
	7x7	300	0	1	0	4066	3495
8 pixel	3x3	150	0	3	0	1803	2050
	5x5	150	0	5	0	4159	6817

Performance Estimate

The following table summarizes the performance of the kernel in different configurations, as generated using Vivado HLS 2019.1 tool for the Xilinx Xczu9eg-ffvb1156-1-i-es1, to process a grayscale HD (1080x1920) image.

Table 401: **Sobel Function Performance Estimate Summary**

Operating Mode	Operating Frequency (MHz)	Filter Size	Latency Estimate (ms)
1 pixel	300	3x3	7.5
	300	5x5	7.5
	300	7x7	7.5
8 pixel	150	3x3	1.7
	150	5x5	1.71

Semi Global Method for Stereo Disparity Estimation

Stereo matching algorithms are used for finding relative depth from a pair of rectified stereo images. The resultant disparity information can be used for 3D reconstruction by triangulation, using the known intrinsic and extrinsic parameters of the stereo camera. The Semi global method for stereo disparity estimation aggregates the cost in terms of dissimilarity across multiple paths leading to a smoother estimate of the disparity map.

For the semi-global method in xfOpenCV, census transform in conjunction with Hamming distance is used for cost computation. The semiglobal optimization block is based on the implementation by Hirschmuller, but approximates the cost aggregation by considering only four directions.

Parallelism is achieved by computing and aggregating cost for multiple disparities in parallel, and this parameter is included as a compile-time input.

API Syntax

```
template<int BORDER_TYPE, int WINDOW_SIZE, int NDISP, int PU, int R, int
SRC_T, int DST_T, int ROWS, int COLS, int NPC>
```

```
void SemiGlobalBM(xf::Mat<SRC_T,ROWS,COLS,NPC> & _src_mat_l,
xf::Mat<SRC_T,ROWS,COLS,NPC> & _src_mat_r, xf::Mat<DST_T,ROWS,COLS,NPC> &
_dst_mat, uint8_t p1, uint8_t p2)
```

Parameter Descriptions

The following table describes the template and the function parameters.

Table 402: **SemiGlobalBM Function Parameter Descriptions**

Parameter	Description
BORDER_TYPE	The border pixels are processed in Census transform function based on this parameter. Only XF_BORDER_CONSTANT is supported.
WINDOW_SIZE	Size of the window used for Census transform computation. Only '5' (5x5) is supported.
NDISP	Number of disparities
PU	Number of disparity units to be computed in parallel
R	Number of directions for cost aggregation. It must be 2, 3, or 4.
SRC_T	Type of input image Mat object. It must be XF_8UC1.
DST_T	Type of output disparity image Mat object. It must be XF_8UC1.
ROWS	Maximum height of the input image.
COLS	Maximum width of the input image.
NPC	Number of pixels to be computed in parallel. It must be XF_NPPC1.
_src_mat_l	Left input image Mat
_src_mat_r	Right input image Mat
_dst_mat	Output disparity image Mat
p1	Small penalty for cost aggregation
p2	Large penalty for cost aggregation. The maximum value is 100.

Resource Utilization

The following table summarizes the resource utilization for a 1920 x 1080 image, with 64 number of disparities, and 32 parallel units.

Table 403: **SemiGlobalBM Function Resource Utilization Summary**

Operating Mode	Filter Size	Operating Frequency (MHz)	Resource Utilization			
			BRAM_18k	DSP48E	FF	LUT
1 pixel	5x5	200	205	141	11856	19102

Performance Estimate

The following table summarizes a performance estimate for a 1920x1080 image.

Table 404: **SemiGlobalBM Function Performance Estimate Summary**

Operating Mode	Operating Frequency	Number of Disparities	Parallel Units	Latency
1 pixel/clock	200 MHz	64	32	42 ms

Stereo Local Block Matching

Stereo block matching is a method to estimate the motion of the blocks between the consecutive frames, called stereo pair. The postulate behind this idea is that, considering a stereo pair, the foreground objects will have disparities higher than the background. Local block matching uses the information in the neighboring patch based on the window size, for identifying the conjugate point in its stereo pair. While, the techniques under global method, used the information from the whole image for computing the matching pixel, providing much better accuracy than local methods. But, the efficiency in the global methods are obtained with the cost of resources, which is where local methods stands out.

Local block matching algorithm consists of pre-processing and disparity estimation stages. The pre-processing consists of Sobel gradient computation followed by image clipping. And the disparity estimation consists of SAD (Sum of Absolute Difference) computation and obtaining the disparity using winner takes all method (least SAD will be the disparity). Invalidity of the pixel relies upon its uniqueness from the other possible disparities. And the invalid pixels are indicated with the disparity value of zero.

API Syntax

```
template <int WSIZE, int NDISP, int NDISP_UNIT, int SRC_T, int DST_T, int
ROWS, int COLS, int NPC = XF_NPPC1,bool USE_URAM=false>
void StereoBM(xf::Mat<SRC_T, ROWS, COLS, NPC> &_left_mat, xf::Mat<SRC_T,
ROWS, COLS, NPC> &_right_mat, xf::Mat<DST_T, ROWS, COLS, NPC> &_disp_mat,
xf::xFSBMState<WSIZE,NDISP,NDISP_UNIT> &sbmstate);
```

Parameter Descriptions

The following table describes the template and the function parameters.

Table 405: **StereoBM Function Parameter Descriptions**

Parameter	Description
WSIZE	Size of the window used for disparity computation
NDISP	Number of disparities
NDISP_UNITS	Number of disparities to be computed in parallel.
SRC_T	Input pixel type. Only 8-bit, unsigned, 1 channel is supported (XF_8UC1)

Table 405: **StereoBM Function Parameter Descriptions** *(cont'd)*

Parameter	Description
DST_T	Output type. This is XF_16UC1, where the disparities are arranged in Q12.4 format.
ROWS	Maximum height of input and output image
COLS	Maximum width of input and output image
NPC	Number of pixels to be processed per cycle; possible options are XF_NPPC1 only.
USE_URAM	Enable to map some storage structures to UltraRAM
left_image	Image from the left camera
right_image	Image from the right camera
disparity_image	Disparities output in the form of an image.
sbmstate	Class object consisting of various parameters regarding the stereo block matching algorithm. 1. preFilterCap: Default value is 31, can be altered by the user, value ranges from 1 to 63 2. minDisparity: Default value is 0, can be altered by the user, value ranges from 0 to (imgWidth-NDISP) 3. uniquenessRatio: Default set to 15, but can be altered to any non-negative integer. 4. textureThreshold: Default set to 10, but can be modified to any non-negative integer.

Resource Utilization

The following table summarizes the resource utilization of the kernel in different configurations, generated using Vivado HLS 2019.1 version tool for the Xilinx® Xczu9eg-ffvb1156-1-i-es1 FPGA, to progress a grayscale HD (1080x1920) image.

The configurations are in the format: imageSize_WSIZE_NDisp_NDispUnits.

Table 406: **StereoBM Function Resource Utilization Summary**

Configurations	Frequency (MHz)	Resource Utilization			
		BRAM_18k	DSP48E	FF	LUT
HD_5_16_2	300	37	20	6856	7181
HD_9_32_4	300	45	20	9700	10396
HD_11_32_32	300	49	20	34519	31978
HD_15_128_32	300	57	20	41017	35176
HD_21_64_16	300	69	20	29853	30706

The following table summarizes the resource utilization of the kernel in different configurations, generated using SDx 2019.1 version tool for the Xilinx xczu7ev-ffvc1156-2-e FPGA, to progress a grayscale HD (1080x1920) image with UltraRAM enable.

The configurations are in the format: imageSize_WSIZE_NDisp_NDispUnits.

Table 407: **StereoBM Function Resource Utilization Summary with UltraRAM Enable**

Configurations	Frequency (MHz)	Resource Utilization				
		BRAM_18k	URAM	DSP48E	FF	LUT
HD_5_16_2	300	0	12	20	7220	6529
HD_9_32_4	300	0	12	20	10186	9302
HD_11_32_32	300	0	14	20	44046	30966
HD_15_128_32	300	0	14	20	50556	38132
HD_21_64_16	300	0	16	20	35991	28464

Performance Estimate

The following table summarizes a performance estimate of the Stereo local block matching in different configurations, as generated using Vivado HLS 2019.1 tool for Xilinx Xczu9eg-ffvb1156-1-i-es1 FPGA, to process a grayscale HD (1080x1920) image.

The configurations are in the format: imageSize_WSIZE_NDisp_NDispUnits.

Table 408: **StereoBM Function Performance Estimate Summary**

Configurations	Frequency (MHz)	Latency (ms)	
		Min	Max
HD_5_16_2	300	55.296	55.296
HD_9_32_4	300	55.296	55.296
HD_11_32_32	300	6.912	6.912
HD_15_48_16	300	20.736	20.736
HD_15_128_32	300	27.648	27.648
HD_21_64_16	300	27.648	27.648

SubRS

The SubRS function subtracts the intensity of the source image from a scalar image and stores it in the destination image.

$$dst(I) = scl - src(I)$$

API Syntax

```
template<int POLICY_TYPE, int SRC_T, int ROWS, int COLS, int NPC =1>
void subRS(xf::Mat<SRC_T, ROWS, COLS, NPC> & _src1, unsigned char
_scl[XF_CHANNELS(SRC_T,NPC)],xf::Mat<SRC_T, ROWS, COLS, NPC> & _dst)
```

Parameter Descriptions

The following table describes the template and the function parameters.

Table 409: **SubRS Function Parameter Descriptions**

Parameter	Description
SRC_T	Input Pixel Type. 8-bit, unsigned, 1 channel is supported (XF_8UC1).
ROWS	Maximum height of input and output image
COLS	Maximum width of input and output image. In case of N-pixel parallelism, width should be multiple of N.
NPC	Number of pixels to be processed per cycle; possible options are XF_NPPC1 and XF_NPPC8 for 1 pixel and 8 pixel operations respectively.
_src1	First Input image
_scl	Input scalar value,the size should be number of channels
_dst	Output image

Resource Utilization

The following table summarizes the resource utilization of the SubRS function in Resource optimized (8 pixel) mode and normal mode as generated using Vivado HLS 2019.1 version tool for the Xczu9eg-ffvb1156-1-i-es1 FPGA.

Table 410: **SubRS Function Resource Utilization Summary**

Name	Resource Utilization	
	1 pixel per clock operation	8 pixel per clock operation
	300 MHz	150 MHz
BRAM_18K	0	0
DSP48E	0	0
FF	103	104
LUT	44	133
CLB	23	43

Performance Estimate

The following table summarizes a performance estimate of the kernel in different configurations, generated using Vivado HLS 2019.1 tool for Xczu9eg-ffvb1156-1-i-es1 FPGA to process a grayscale HD (1080x1920) image.

Table 411: **SubRS Function Performance Estimate Summary**

Operating Mode	Latency Estimate	
	Operating Frequency (MHz)	Latency (ms)
1 pixel	300	6.9
8 pixel	150	1.7

SubS

The SubS function subtracts a scalar value from the intensity of source image and stores it in the destination image.

$$dst(I)= src(I) - scl$$

API Syntax

```
template<int POLICY_TYPE, int SRC_T, int ROWS, int COLS, int NPC =1>
void subS(xf::Mat<SRC_T, ROWS, COLS, NPC> & _src1, unsigned char
_scl[XF_CHANNELS(SRC_T,NPC)],xf::Mat<SRC_T, ROWS, COLS, NPC> & _dst)
```

Parameter Descriptions

The following table describes the template and the function parameters.

Table 412: **SubS Function Parameter Descriptions**

Parameter	Description
SRC_T	Input Pixel Type. 8-bit, unsigned, 1 channel is supported (XF_8UC1).
ROWS	Maximum height of input and output image.
COLS	Maximum width of input and output image. In case of N-pixel parallelism, width should be multiple of N.
NPC	Number of pixels to be processed per cycle; possible options are XF_NPPC1 and XF_NPPC8 for 1 pixel and 8 pixel operations respectively.
_src1	First Input image
_scl	Input scalar value, the size should be the number of channels.
_dst	Output image

Resource Utilization

The following table summarizes the resource utilization of the SubS function in Resource optimized (8 pixel) mode and normal mode as generated using Vivado HLS 2019.1 version tool for the Xczu9eg-ffvb1156-1-i-es1 FPGA.

Table 413: **SubS Function Resource Utilization Summary**

Name	Resource Utilization	
	1 pixel per clock operation	8 pixel per clock operation
	300 MHz	150 MHz
BRAM_18K	0	0
DSP48E	0	0
FF	103	104
LUT	44	133
CLB	23	43

Performance Estimate

The following table summarizes a performance estimate of the kernel in different configurations, generated using Vivado HLS 2019.1 tool for Xczu9eg-ffvb1156-1-i-es1 FPGA to process a grayscale HD (1080x1920) image.

Table 414: **SubS Function Performance Estimate Summary**

Operating Mode	Latency Estimate	
	Operating Frequency (MHz)	Latency (ms)
1 pixel	300	6.9
8 pixel	150	1.7

Sum

The sum function calculates the sum of all pixels in input image.

API Syntax

```
template< int SRC_T , int ROWS, int COLS, int NPC=1>
void sum(xf::Mat<SRC_T, ROWS, COLS, NPC> & src1,double
sum[XF_CHANNELS(SRC_T,NPC)])
```

Parameter Descriptions

The following table describes the template and the function parameters.

Table 415: **Sum Function Parameter Descriptions**

Parameter	Description
SRC_T	Input pixel type. 8-bit, unsigned, 1 channel is supported (XF_8UC1).
ROWS	Maximum height of input and output image.
COLS	Maximum width of input and output image (must be multiple of 8).
NPC	Number of pixels to be processed per cycle.
_src1	Input image.
sum	Array to store sum of all pixels in the image.

Resource Utilization

The following table summarizes the resource utilization of the Sum function in Resource optimized (8 pixel) mode and normal mode as generated using Vivado HLS 2019.1 version tool for the Xczu9eg-ffvb1156-1-i-es1 FPGA.

Table 416: **Sum Function Resource Utilization Summary**

Name	Resource Utilization	
	1 pixel per clock operation	8 pixel per clock operation
	300 MHz	150 MHz
BRAM_18K	0	0
DSP48E	0	0
FF	341	408
LUT	304	338
CLB	71	87

Performance Estimate

The following table summarizes a performance estimate of the kernel in different configurations, generated using Vivado HLS 2019.1 tool for Xczu9eg-ffvb1156-1-i-es1 FPGA to process a grayscale HD (1080x1920) image.

Table 417: **Sum Function Performance Estimate Summary**

Operating Mode	Latency Estimate	
	Operating Frequency (MHz)	Latency (ms)
1 pixel	300	
8 pixel	150	

SVM

The `SVM` function is the SVM core operation, which performs dot product between the input arrays. The function returns the resultant dot product value with its fixed point type.

API Syntax

```
template<int SRC1_T, int SRC2_T, int DST_T, int ROWS1, int COLS1, int
ROWS2, int COLS2, int NPC=1, int N>
void SVM(xf::Mat<SRC1_T, ROWS1, COLS1, NPC> &in_1, xf::Mat<SRC2_T, ROWS2,
COLS2, NPC> &in_2, uint16_t idx1, uint16_t idx2, uchar_t frac1, uchar_t
frac2, uint16_t n, uchar_t *out_frac, ap_int<XF_PIXELDEPTH(DST_T)> *result)
```

Parameter Descriptions

The following table describes the template and the function parameters.

Table 418: **SVM Function Parameter Descriptions**

Parameters	Description
SRC1_T	Input pixel type. 16-bit, signed, 1 channel (XF_16SC1) is supported.
SRC2_T	Input pixel type. 16-bit, signed, 1 channel (XF_16SC1) is supported.

Table 418: **SVM Function Parameter Descriptions** *(cont'd)*

Parameters	Description
DST_T	Output data Type. 32-bit, signed, 1 channel (XF_32SC1) is supported.
ROWS1	Number of rows in the first image being processed.
COLS1	Number of columns in the first image being processed.
ROWS2	Number of rows in the second image being processed.
COLS2	Number of columns in the second image being processed.
NPC	Number of pixels to be processed per cycle; possible options are XF_NPPC1.
N	Max number of kernel operations
in_1	First Input Array.
in_2	Second Input Array.
idx1	Starting index of the first array.
idx2	Starting index of the second array.
frac1	Number of fractional bits in the first array data.
frac2	Number of fractional bits in the second array data.
n	Number of kernel operations.
out_frac	Number of fractional bits in the resultant value.
result	Resultant value

Resource Utilization

The following table summarizes the resource utilization of the SVM function, generated using Vivado HLS 2019.1 tool for the Xilinx Xczu9eg-ffvb1156-1-i-es1 FPGA.

Table 419: **SVM Function Resource Utilization Summary**

Operating Frequency (MHz)	Utilization Estimate (ms)				
	BRAM_18K	DSP_48Es	FF	LUT	CLB
300	0	1	27	34	12

Performance Estimate

The following table summarizes the performance in different configurations, as generated using Vivado HLS 2019.1 tool for the Xilinx Xczu9eg-ffvb1156-1-i-es1 FPGA.

Table 420: **SVM Function Performance Estimate Summary**

Operating Frequency (MHz)	Latency Estimate	
	Min (cycles)	Max (cycles)
300	204	204

Thresholding

The `Threshold` function performs thresholding operation on the input image. There are several types of thresholding supported by the function.

$$dst(x, y) = \begin{cases} maxval, & if \;\; src(x, y) > threshold \\ 0, & Otherwise \end{cases}$$

$$dst(x, y) = \begin{cases} 0, & if \;\; src(x, y) > threshold \\ maxval, & Otherwise \end{cases}$$

$$dst(x, y) = \begin{cases} threshold, & if \;\; src(x, y) > threshold \\ src(x, y), & Otherwise \end{cases}$$

$$dst(x, y) = \begin{cases} src(x, y), & if \;\; src(x, y) > threshold \\ 0, & Otherwise \end{cases}$$

$$dst(x, y) = \begin{cases} 0, & if \;\; src(x, y) > threshold \\ src(x, y), & Otherwise \end{cases}$$

API Syntax

```
template<int THRESHOLD_TYPE, int SRC_T, int ROWS, int COLS,int NPC=1>
void Threshold(xf::Mat<SRC_T, ROWS, COLS, NPC> & _src_mat,xf::Mat<SRC_T,
ROWS, COLS, NPC> & _dst_mat,short int thresh,short int maxval )
```

Parameter Descriptions

The following table describes the template and the function parameters.

Table 421: **Threshold Function Parameter Descriptions**

Parameter	Description
THRESHOLD_TYPE	Type of thresholding.
SRC_T	Input pixel type. Only 8-bit, unsigned, 1 channel is supported (XF_8UC1).
ROWS	Maximum height of input and output image.
COLS	Maximum width of input and output image. Must be multiple of 8, for 8-pixel operation.
NPC	Number of pixels to be processed per cycle.
_src_mat	Input image
_dst_mat	Output image
thresh	Threshold value.
maxval	Maximum value to use with the `THRESH_BINARY` and `THRESH_BINARY_INV` thresholding types.

Resource Utilization

The following table summarizes the resource utilization of the kernel with binary thresholding in different configurations, generated using Vivado HLS 2019.1 tool for the Xilinx Xczu9eg-ffvb1156-1 FPGA, to process a grayscale HD (1080x1920) image.

Table 422: **Threshold Function Resource Utilization Summary**

Configurations	Resource Utilization	
	1 pixel	8 pixel
	300 MHz	150 MHz
BRAM_18K	0	0
DSP48E	0	0
FF	110	154
LUT	61	139
CLB	16	37

Performance Estimate

The following table summarizes the performance of the kernel in different configurations, as generated using Vivado HLS 2019.1 tool for the Xilinx Xczu9eg-ffvb1156-1, to process a grayscale HD (1080x1920) image.

Table 423: **Threshold Function Performance Estimate Summary**

Operating Mode	Operating Frequency (MHz)	Latency Estimate (ms)
1 pixel	300	7.2
8 pixel	150	1.7

Atan2

The `Atan2LookupFP` function finds the arctangent of y/x. It returns the angle made by the vector $\begin{bmatrix} x \\ y \end{bmatrix}$ with respect to origin. The angle returned by atan2 will also contain the quadrant information.

`Atan2LookupFP` is a fixed point version of the standard atan2 function. This function implements the atan2 using a lookup table approach. The values in the look up table are represented in Q4.12 format and so the values returned by this function are in Q4.12. A maximum error of 0.2 degrees is present in the range of 89 to 90 degrees when compared to the standard atan2 function available in glibc. For the other angles (0 to 89) the maximum error is in the order of 10-3. This function returns 0 when both xs and ys are zeroes.

API Syntax

```
short Atan2LookupFP(short xs, short ys, int M1,int N1,int M2, int N2)
```

Parameter Descriptions

The following table describes the template and the function parameters.

Table 424: **Atan2LookupFP Function Parameter Descriptions**

Parameter	Description
xs	16-bit signed value x in fixed point format of QM1.N1
ys	16-bit signed value y in fixed point format of QM2.N2
M1	Number of bits to represent integer part of x.
N1	Number of bits to represent fractional part of y. Must be equal to 16-M1.
M2	Number of bits to represent integer part of y.
N2	Number of bits to represent fractional part of y. Must be equal to 16-N1.
Return	Return value is in radians. Its range varies from -pi to +pi in fixed point format of Q4.12

Resource Utilization

The following table summarizes the resource utilization of the `Atan2LookupFP` function , generated using Vivado HLS 2019.1 tool for the Xilinx Xczu9eg-ffvb1156-1-i-es1 FPGA.

Table 425: **Atan2LookupFP Function Resource Utilization Summary**

Operating Frequency (MHz)	Utilization Estimate				
	BRAM_18K	DSP_48Es	FF	LUT	CLB
300	4	2	275	75	139

Performance Estimate

The following table summarizes the performance in different configurations, as generated using Vivado HLS 2019.1 tool for the Xilinx Xczu9eg-ffvb1156-1-i-es1 FPGA.

Table 426: **Atan2LookupFP Function Performance Estimate Summary**

Operating Frequency (MHz)	Latency Estimate	
	Min (cycles)	Max (cycles)
300	1	15

Inverse (Reciprocal)

The `Inverse` function computes the reciprocal of a number x. The values of 1/x are stored in a look up table of 2048 size. The index for picking the 1/x value is computed using the fixed point format of x. Once this index is computed, the corresponding 1/x value is fetched from the look up table and returned along with the number of fractional bits needed to represent this value in fixed point format.

API Syntax

```
unsigned int Inverse(unsigned short x,int M,char *N)
```

Parameter Descriptions

The following table describes the template and the function parameters.

Table 427: **Inverse Function Parameter Descriptions**

Parameter	Description
x	16-bit unsigned value x in fixed point format of QM.(16-M)
M	Number of bits to represent integer part of x.
N	Pointer to a char variable which stores the number of bits to represent fractional part of 1/x. This value is returned from the function.
Return	1/x value is returned in 32-bit format represented by a fixed point format of Q(32-N).N

Resource Utilization

The following table summarizes the resource utilization of the Inverse function, generated using Vivado HLS 2019.1 tool for the Xilinx Xczu9eg-ffvb1156-1-i-es1 FPGA.

Table 428: **Inverse Function Resource Utilization Summary**

Operating Frequency (MHz)	Utilization Estimate (ms)				
	BRAM_18K	DSP_48Es	FF	LUT	CLB
300	4	0	68	128	22

Performance Estimate

The following table summarizes the performance in different configurations, as generated using Vivado HLS 2019.1 tool for the Xilinx Xczu9eg-ffvb1156-1-i-es1 FPGA.

Table 429: **Inverse Function Performance Estimate Summary**

Operating Frequency (MHz)	Latency Estimate	
	Min (cycles)	Max (cycles)
300	1	8

Look Up Table

The LUT function performs the table lookup operation. Transforms the source image into the destination image using the given look-up table. The input image must be of depth XF_8UP and the output image of same type as input image.

$I_{out}(x, y) = LUT\ [I_{in1}(x, y)]$

Where:

- $I_{out}(x, y)$ is the intensity of output image at (x, y) position
- $I_{in}(x, y)$ is the intensity of first input image at (x, y) position
- LUT is the lookup table of size 256 and type unsigned char.

API Syntax

```
template <int SRC_T, int ROWS, int COLS,int NPC=1>
void LUT(xf::Mat<SRC_T, ROWS, COLS, NPC> & _src, xf::Mat<SRC_T, ROWS, COLS,
NPC> & _dst,unsigned char* _lut)
```

Parameter Descriptions

The following table describes the template and the function parameters.

Table 430: **LUT Function Parameter Descriptions**

Parameter	Description
SRC_T	Input and Output pixel type. Only 8-bit, unsigned, 1 and 3 channels are supported (XF_8UC1 and XF_8UC3)
ROWS	Number of rows in the image being processed.
COLS	Number of columns in the image being processed. Must be a multiple of 8, for 8-pixel operation.
NPC	Number of pixels to be processed in parallel. Possible options are XF_NPPC1 and XF_NPPC8 for 1 pixel and 8 pixel operations respectively.
_src	Input image of size (ROWS, COLS) and type 8U.
_dst	Output image of size (ROWS, COLS) and same type as input.
_lut	Input lookup Table of size 256 and type unsigned char.

Resource Utilization

The following table summarizes the resource utilization of the LUT function, generated using Vivado HLS 2019.1 tool for the Xilinx Xczu9eg-ffvb1156-1-i-es1 FPGA, to process a grayscale HD (1080x1920) image.

Table 431: **LUT Function Resource Utilization Summary**

Operating Mode	Operating Frequency (MHz)	Utilization Estimate				
		BRAM_18K	DSP_48Es	FF	LUT	CLB
1 pixel	300	1	0	937	565	137
8 pixel	150	9	0	1109	679	162

The following table summarizes the resource utilization of the LUT function, generated using Vivado HLS 2019.1 tool for the Xilinx Xczu9eg-ffvb1156-1-i-es1 FPGA, to process 4K 3Channel image.

Table 432: **LUT Function Resource Utilization Summary**

Operating Mode	Operating Frequency (MHz)	Utilization Estimate				
		BRAM_18K	DSP_48Es	FF	LUT	CLB
1 pixel	300	4	0	1160	648	175

Performance Estimate

The following table summarizes the performance in different configurations, as generated using Vivado HLS 2019.1 tool for the Xilinx Xczu9eg-ffvb1156-1-i-es1, to process a grayscale HD (1080x1920) image.

Table 433: **LUT Function Performance Estimate Summary**

Operating Mode	Latency Estimate
	Max Latency
1 pixel operation (300 MHz)	6.92 ms
8 pixel operation (150 MHz)	1.66 ms

Square Root

The `Sqrt` function computes the square root of a 16-bit fixed point number using the non-restoring square root algorithm. The non-restoring square root algorithm uses the two's complement representation for the square root result. At each iteration the algorithm can generate exact result value even in the last bit.

Input argument D must be 16-bit number, though it is declared as 32-bit. The output sqrt(D) is 16-bit type. If format of D is QM.N (where M+N = 16) then format of output is Q(M/2).N

To get a precision of 'n' bits in fractional part, you can simply left shift the radicand (D) by '2n' before the function call and shift the solution right by 'n' to get the correct answer. For example, to find the square root of 35 (01100011_2) with one bit after the decimal point, that is, N=1:

1. Shift the number (0110001100_2) left by 2
2. Shift the answer (1011_2) right by 1. The correct answer is 101.1, which is 5.5.

API Syntax

```
int Sqrt(unsigned int D)
```

Parameter Descriptions

The following table describes the template and the function parameters.

Table 434: **Sqrt Function Parameter Descriptions**

Parameter	Description
D	Input data in a 16-bit fixed-point format.
Return	Output value in short int format.

Resource Utilization

The following table summarizes the resource utilization of the Sqrt function, generated using Vivado HLS 2019.1 tool for the Xilinx Xczu9eg-ffvb1156-1-i-es1 FPGA.

Table 435: **Sqrt Function Resource Utilization Summary**

Operating Frequency (MHz)	Utilization Estimate				
	BRAM_18K	DSP_48Es	FF	LUT	CLB
300	0	0	8	6	1

Performance Estimate

The following table summarizes the performance in different configurations, as generated using Vivado HLS 2019.1 tool for the Xilinx Xczu9eg-ffvb1156-1-i-es1 FPGA.

Table 436: **Sqrt Function Performance Estimate Summary**

Operating Frequency (MHz)	Latency Estimate	
	Min (cycles)	Max (cycles)
300	18	18

WarpTransform

The `warpTransform` function is designed to perform the perspective and affine geometric transformations on an image. The type of transform is a compile time parameter to the function.

The function uses a streaming interface to perform the transformation. Due to this and due to the fact that geometric transformations need access to many different rows of input data to compute one output row, the function stores some rows of the input data in block RAMs/ UltraRAMs. The number of rows the function stores can be configured by the user by modifying a template parameter. Based on the transformation matrix, you can decide on the number of rows to be stored. You can also choose when to start transforming the input image in terms of the number of rows of stored image.

Affine Transformation

The transformation matrix consists of size parameters, and is as shown:

$$M = \begin{bmatrix} M_{11} & M_{12} & M_{13} \\ M_{21} & M_{22} & M_{23} \end{bmatrix}$$

Affine transformation is applied in the warpTransform function following the equation:

$$dst\begin{pmatrix} x \\ y \end{pmatrix} = M * src\begin{pmatrix} x \\ y \\ 1 \end{pmatrix}$$

Perspective Transformation

The transformation matrix is a 3x3 matrix as shown below:

$$M = \begin{bmatrix} M_{11} & M_{12} & M_{13} \\ M_{21} & M_{22} & M_{23} \\ M_{31} & M_{32} & M_{33} \end{bmatrix}$$

Perspective transformation is applied in warpTransform following the equation:

$$dst^1\begin{pmatrix} x \\ y \\ n \end{pmatrix} = M * src\begin{pmatrix} x \\ y \\ 1 \end{pmatrix}$$

The destination pixel is then computed by dividing the first two dimensions of the dst1 by the third dimension

$$dst^1\begin{pmatrix} x \\ y \\ n \end{pmatrix} = M * src\begin{pmatrix} x \\ y \\ 1 \end{pmatrix}$$

API Syntax

```
template<int STORE_LINES, int START_ROW, int TRANSFORMATION_TYPE, int
INTERPOLATION_TYPE, int SRC_T, int ROWS, int COLS, int NPC=1,bool
USE_URAM=false>
void warpTransform(xf::Mat<SRC_T, ROWS, COLS, NPC> & src, xf::Mat<SRC_T,
ROWS, COLS, NPC> & dst, float *transformation_matrix)
```

Parameter Descriptions

The following table describes the template and the function parameters.

Table 437: **warpTransform Function Parameter Descriptions**

Parameter	Description
STORE_LINES	Number of lines to store an input to process a given transformation.
START_ROW	Number of the input rows to store before starting the image transformation. This must be less than or equal to STORE_LINES.
TRANSFORMATION_TYPE	Affine and perspective transformations are supported. Set this flag to '0' for affine and '1' for perspective transformation.
INTERPOLATION_TYPE	Set flag to '1' for bilinear interpolation and '0' for nearest neighbor interpolation.
SRC_T	Input and Output pixel type. Only 8-bit, unsigned, 1 and 3 channels are supported (XF_8UC1 and XF_8UC3)
ROWS	Maximum height of input and output image.
COLS	Maximum width of input and output image.
NPC	Number of pixels to be processed per cycle; only one-pixel operation supported (XF_NPPC1).
USE_URAM	Enable to map some storage structures to UltraRAM
src	Input image
dst	Output image
transformation_matrix	Transformation matrix that is applied to the input image.

Resource Utilization

The following table summarizes the resource utilization of the Warp transform, generated using Vivado HLS 2019.1 version tool for the Xilinx Number of lines of the image that need to be buffered locally on FPGA.Xczu9eg-ffvb1156-1-i-es1 FPGA, to process a grayscale HD (1080x1920) image.

Table 438: **warpTransform Function Resource Utilization Summary**

Transformation	INTERPOLATION _TYPE	STORE _LINES	START _ROW	Operating Frequency (MHz)	Utilization Estimate			
					LUTs	FFs	DSPs	BRAMs
Perspective	Bilinear	100	50	300	7468	9804	61	112
Perspective	Nearest Neighbor	100	50	300	4514	6761	35	104
Affine	Bilinear	100	50	300	6139	5606	40	124
Affine	Nearest Neighbor	100	50	300	4611	4589	18	112

Number of lines of the image that need to be buffered locallyThe following table summarizes the resource utilization of the Warp transform, generated using Vivado HLS 2019.1 version tool for the Xilinx Xczu9eg-ffvb1156-1-i-es1 FPGA, to process a BGR 4K image.

Table 439: **warpTransform Function Resource Utilization Summary**

Transformation	INTERPOLATION _TYPE	STORE _LINES	START _ROW	Operating Frequency (MHz)	Utilization Estimate			
					LUTs	FFs	DSPs	BRAMs
Perspective	Bilinear	100	50	300	9192	7910	48	616
Perspective	Nearest Neighbor	100	50	300	10533	12055	69	604
Affine	Bilinear	100	50	300	6397	8415	35	604

The following table summarizes the resource utilization of the Warp transform, generated using SDx 2019.1 version tool for the Xilinx xczu7ev-ffvc1156-2-e FPGA, to progress a grayscale 4K image with UltraRAM enabled.

Table 440: **warpTransform Function Resource Utilization Summary with UltraRAM Enable**

Transformation	INTERPOLATION _TYPE	STORE _LINES	START _ROW	Operatin g Frequenc y (MHz)	Utilization Estimate				
					LUTs	FFs	DSPs	BRAMs	URAM
Perspective	Bilinear	100	50	300	7820	12458	61	7	12
Perspective	Nearest Neighbor	100	50	300	4880	8323	35	2	6
Affine	Bilinear	100	50	300	6850	9516	40	13	12
Affine	Nearest Neighbor	100	50	300	4651	6548	18	6	6

Performance Estimate

The following table summarizes a performance estimate of the Warp transform, as generated using Vivado HLS 2019.1 tool for Xilinx Xczu9eg-ffvb1156-1-i-es1 FPGA, to process a grayscale HD (1080x1920) image.

Table 441: **warpTransform Function Performance Estimate Summary**

Transformation	INTERPOLATION _TYPE	STORE _LINES	START _ROW	Operating Frequency (MHz)	Latency Estimate Max (ms)
Perspective	Bilinear	100	50	300	7.46
Perspective	Nearest Neighbor	100	50	300	7.31
Affine	Bilinear	100	50	300	7.31
Affine	Nearest Neighbor	100	50	300	7.24

Zero

The Zero function sets the each pixel in input image to zero and stores the result in dst.

API Syntax

```
template< int SRC_T , int ROWS, int COLS, int NPC=1>
void zero(xf::Mat<SRC_T, ROWS, COLS, NPC> & _src1,xf::Mat<SRC_T, ROWS,
COLS, NPC> & _dst)
```

Parameter Descriptions

The following table describes the template and the function parameters.

Table 442: **Zero Function Parameter Descriptions**

Parameter	Description
SRC_T	Input Pixel Type. 8-bit, unsigned, 1 channel is supported (XF_8UC1).
ROWS	Maximum height of input and output image.
COLS	Maximum width of input and output image. In case of N-pixel parallelism, width should be multiple of N.
NPC	Number of pixels to be processed per cycle; possible options are XF_NPPC1 and XF_NPPC8 for 1 pixel and 8 pixel operations respectively.
_src1	Input image
_dst	Output image

Resource Utilization

The following table summarizes the resource utilization of the Zero function in Resource optimized (8 pixel) mode and normal mode as generated using Vivado HLS 2019.1 version tool for the Xczu9eg-ffvb1156-1-i-es1 FPGA.

Table 443: **Zero Function Resource Utilization Summary**

Name	Resource Utilization	
	1 pixel per clock operation	8 pixel per clock operation
	300 MHz	150 MHz
BRAM_18K	0	0
DSP48E	0	0
FF	78	78
LUT	42	41
CLB	15	14

Performance Estimate

The following table summarizes a performance estimate of the kernel in different configurations, generated using Vivado HLS 2019.1 tool for Xczu9eg-ffvb1156-1-i-es1 FPGA to process a grayscale HD (1080x1920) image.

Table 444: **Zero Function Performance Estimate Summary**

Operating Mode	Latency Estimate	
	Operating Frequency (MHz)	Latency (ms)
1 pixel	300	6.9
8 pixel	150	1.7

Design Examples Using xfOpenCV Library

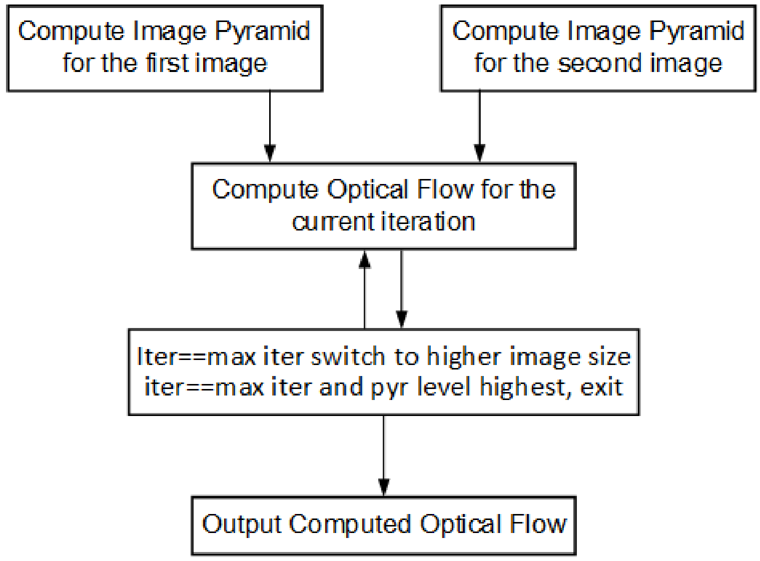
All the hardware functions in the library have their own respective examples that are available in the github. This section provides details of image processing functions and pipelines implemented using a combination of various functions in xfOpenCV. They illustrate how to best implement various functionalities using the capabilities of both the processor and the programmable logic. These examples also illustrate different ways to implement complex dataflow paths. The following examples are described in this section:

- Iterative Pyramidal Dense Optical Flow
- Corner Tracking Using Sparse Optical Flow
- Color Detection
- Difference of Gaussian Filter
- Stereo Vision Pipeline

Iterative Pyramidal Dense Optical Flow

The Dense Pyramidal Optical Flow example uses the `xf::pyrDown` and `xf::densePyrOpticalFlow` hardware functions from the xfOpenCV library, to create an image pyramid, iterate over it and compute the Optical Flow between two input images. The example uses two hardware instances of the `xf::pyrDown` function to compute the image pyramids of the two input images in parallel. The two image pyramids are processed by one hardware instance of the `xf::densePyrOpticalFlow` function, starting from the smallest image size going up to the largest image size. The output flow vectors of each iteration are fed back to the hardware kernel as input to the hardware function. The output of the last iteration on the largest image size is treated as the output of the dense pyramidal optical flow example.

Figure 11: **Iterative Pyramidal Dense Optical Flow**



Specific details of the implementation of the example on the host follow to help understand the process in which the claimed throughput is achieved.

pyrof_hw()

The `pyrof_hw()` is the host function that computes the dense optical flow.

API Syntax

```
void pyrof_hw(cv::Mat im0, cv::Mat im1, cv::Mat flowUmat, cv::Mat flowVmat,
xf::Mat<XF_32UC1,HEIGHT,WIDTH,XF_NPPC1> & flow,
xf::Mat<XF_32UC1,HEIGHT,WIDTH,XF_NPPC1> & flow_iter,
xf::Mat<XF_8UC1,HEIGHT,WIDTH,XF_NPPC1> mat_imagepyr1[NUM_LEVELS] ,
xf::Mat<XF_8UC1,HEIGHT,WIDTH,XF_NPPC1> mat_imagepyr2[NUM_LEVELS] , int
pyr_h[NUM_LEVELS], int pyr_w[NUM_LEVELS])
```

Parameter Descriptions

The table below describes the template and the function parameters.

Parameter	Description
im0	First input image in cv::Mat
im1	Second input image in cv::Mat
flowUmat	Allocated cv::Mat to store the horizontal component of the output flow vector
flowVmat	Allocated cv::Mat to store the vertical component of the output flow vector
flow	Allocated xf::Mat to temporarily store the packed flow vectors, during the iterative computation using the hardware function
flow_iter	Allocated xf::Mat to temporarily store the packed flow vectors, during the iterative computation using the hardware function

Parameter	Description
mat_imagepyr1	An array, of size equal to the number of image pyramid levels, of xf::Mat to store the image pyramid of the first image
mat_imagepyr2	An array, of size equal to the number of image pyramid levels, of xf::Mat to store the image pyramid of the second image
pyr_h	An array of integers which includes the size of number of image pyramid levels, to store the height of the image at each pyramid level
pyr_w	An array of integers which includes the size of the number of image pyramid levels, to store the width of the image at each pyramid level

Dataflow

The `pyrof_hw()` function performs the following:

1. Set the sizes of the images in various levels of the image pyramid
2. Copy input images from cv::Mat format to the xf::Mat object allocated to contain the largest image pyramid level
3. Create the image pyramid calling the `pyr_dense_optical_flow_pyr_down_accel()` function
4. Use the `pyr_dense_optical_flow_accel()` function to compute the optical flow output by iterating over the pyramid levels as input by the user
5. Unpack the flow vectors and convert them to the floating point, and return

The important steps 3 and 4 in the above processes will be explained in detail.

pyr_dense_optical_flow_pyr_down_accel()

API Syntax

```
void
pyr_dense_optical_flow_pyr_down_accel(xf::Mat<XF_8UC1,HEIGHT,WIDTH,XF_NPPC1>
 mat_imagepyr1[NUM_LEVELS], xf::Mat<XF_8UC1,HEIGHT,WIDTH,XF_NPPC1>
mat_imagepyr2[NUM_LEVELS])
```

Parameter Descriptions

The table below describes the template and the function parameters.

Parameter	Description
mat_imagepyr1	An array, of size equal to the number of image pyramid levels, of xf::Mat to store the image pyramid of the first image. The memory location corresponding to the highest pyramid level [0] in this allocated memory must contain the first input image.
mat_imagepyr2	An array, of size equal to the number of image pyramid levels, of xf::Mat to store the image pyramid of the second image. The memory location corresponding to the highest pyramid level [0] in this allocated memory must contain the second input image.

The `pyr_dense_optical_flow_pyr_down_accel()` just runs one for loop calling the `xf::pyrDown` hardware function as follows:

```
for(int pyr_comp=0;pyr_comp<NUM_LEVELS-1; pyr_comp++)
    {
    #pragma SDS async(1)
    #pragma SDS resource(1)

xf::pyrDown<XF_8UC1,HEIGHT,WIDTH,XF_NPPC1,XF_USE_URAM>(mat_imagepyr1[pyr_com
p], mat_imagepyr1[pyr_comp+1]);
    #pragma SDS async(2)
    #pragma SDS resource(2)

xf::pyrDown<XF_8UC1,HEIGHT,WIDTH,XF_NPPC1,XF_USE_URAM>(mat_imagepyr2[pyr_com
p], mat_imagepyr2[pyr_comp+1]);
    #pragma SDS wait(1)
    #pragma SDS wait(2)
    }
```

The code is straightforward without the pragmas, and the `xf::pyrDown` function is being called twice every iteration. First with the first image and then with the second image. Note that the input to the next iteration is the output of the current iteration. The pragma #pragma SDS async(ID) makes the Arm® processor call the hardware function and not wait for the hardware function to return. The Arm processor takes some cycles to call the function, which includes programming the DMA. The pragma #pragma SDS wait(ID) makes the Arm processor wait for the hardware function called with the async(ID) pragma to finish processing. The pragma #pragma SDS resource(ID) creates a separate hardware instance each time the hardware function is called with a different ID. With this new information it is easy to assimilate that the loop in the above host function calls the two hardware instances of `xf::pyrDown` functions in parallel, waits until both the functions return and proceed to the next iteration.

Dense Pyramidal Optical Flow Computation

```
for (int l=NUM_LEVELS-1; l>=0; l--) {
        //compute current level height
        int curr_height = pyr_h[l];
        int curr_width = pyr_w[l];

        //compute the flow vectors for the current pyramid level iteratively
        for(int iterations=0;iterations<NUM_ITERATIONS; iterations++)
        {
            bool scale_up_flag = (iterations==0)&&(l != NUM_LEVELS-1);
            int next_height = (scale_up_flag==1)?pyr_h[l+1]:pyr_h[l];
            int next_width  = (scale_up_flag==1)?pyr_w[l+1]:pyr_w[l];
            float scale_in = (next_height - 1)*1.0/(curr_height - 1);
            ap_uint<1> init_flag = ((iterations==0) && (l==NUM_LEVELS-1))?
1 : 0;
            if(flag_flowin)
            {
                flow.rows = pyr_h[l];
                flow.cols = pyr_w[l];
                flow.size = pyr_h[l]*pyr_w[l];
                pyr_dense_optical_flow_accel(mat_imagepyr1[l],
mat_imagepyr2[l], flow_iter, flow, l, scale_up_flag, scale_in, init_flag);
                flag_flowin = 0;
            }
```

```
            else
            {
                flow_iter.rows = pyr_h[l];
                flow_iter.cols = pyr_w[l];
                flow_iter.size = pyr_h[l]*pyr_w[l];
                pyr_dense_optical_flow_accel(mat_imagepyr1[l],
mat_imagepyr2[l], flow, flow_iter, l, scale_up_flag, scale_in, init_flag);
                flag_flowin = 1;
            }
        }//end iterative coptical flow computation
    } // end pyramidal iterative optical flow HLS computation
```

The Iterative Pyramidal Dense Optical Flow is computed in a nested for loop which runs for iterations*pyramid levels number of iterations. The main loop starts from the smallest image size and iterates up to the largest image size. Before the loop iterates in one pyramid level, it sets the current pyramid level's height and width, in curr_height and current_width variables. In the nested loop, the next_height variable is set to the previous image height if scaling up is necessary, that is, in the first iterations. As divisions are costly and one time divisions can be avoided in hardware, the scale factor is computed in the host and passed as an argument to the hardware kernel. After each pyramid level, in the first iteration, the scale-up flag is set to let the hardware function know that the input flow vectors need to be scaled up to the next higher image size. Scaling up is done using bilinear interpolation in the hardware kernel.

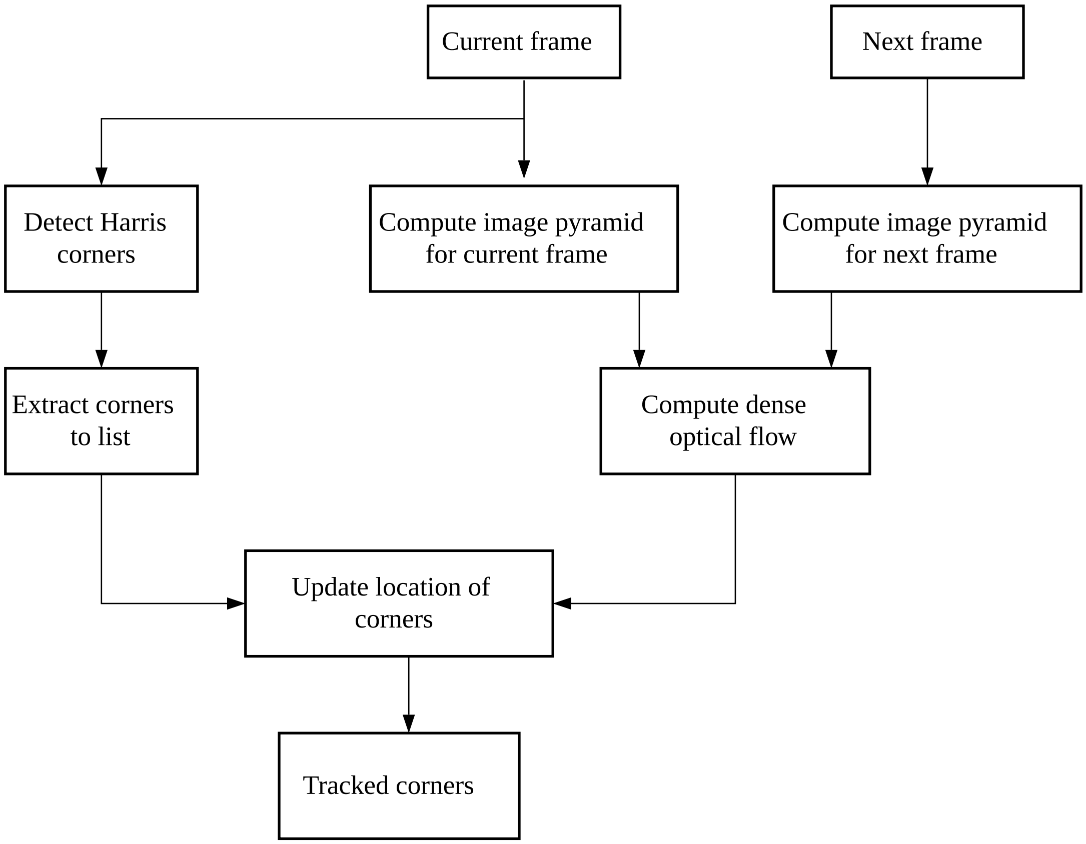
After all the input data is prepared, and the flags are set, the host processor calls the hardware function. Please note that the host function swaps the flow vector inputs and outputs to the hardware function to iteratively solve the optimization problem. Also note that the `pyr_dense_optical_flow_accel()` function is just a wrapper to the hardware function `xf::densePyrOpticalFlow`. Template parameters to the hardware function are passed inside this wrapper function.

Corner Tracking Using Sparse Optical Flow

This example illustrates how to detect and track the characteristic feature points in a set of successive frames of video. A Harris corner detector is used as the feature detector, and a modified version of Lucas Kanade optical flow is used for tracking. The core part of the algorithm takes in current and next frame as the inputs and outputs the list of tracked corners. The current image is the first frame in the set, then corner detection is performed to detect the features to track. The number of frames in which the points need to be tracked is also provided as the input.

Corner tracking example uses five hardware functions from the xfOpenCV library `xf::cornerHarris`, `xf:: cornersImgToList`, `xf::cornerUpdate`, `xf::pyrDown`, and `xf::densePyrOpticalFlow`.

Figure 12: **Corner Tracking Using Sparse Optical Flow**



A new hardware function, `xf::cornerUpdate`, has been added to ensure that the dense flow vectors from the output of the`xf::densePyrOpticalFlow` function are sparsely picked and stored in a new memory location as a sparse array. This was done to ensure that the next function in the pipeline would not have to surf through the memory by random accesses. The function takes corners from Harris corner detector and dense optical flow vectors from the dense pyramidal optical flow function and outputs the updated corner locations, tracking the input corners using the dense flow vectors, thereby imitating the sparse optical flow behavior. This hardware function runs at 300 MHz for 10,000 corners on a 720p image, adding very minimal latency to the pipeline.

cornerUpdate()

API Syntax

```
template <unsigned int MAXCORNERSNO, unsigned int TYPE, unsigned int ROWS,
unsigned int COLS, unsigned int NPC>
void cornerUpdate(ap_uint<64> *list_fix, unsigned int *list, uint32_t
nCorners, xf::Mat<TYPE,ROWS,COLS,NPC> &flow_vectors, ap_uint<1> harris_flag)
```

Parameter Descriptions

The following table describes the template and the function parameters.

Table 445: **CornerUpdate Function Parameter Descriptions**

Parameter	Description
MAXCORNERSNO	Maximum number of corners that the function needs to work on
TYPE	Input Pixel Type. Only 8-bit, unsigned, 1 channel is supported (XF_8UC1)
ROWS	Maximum height of input and output image (Must be multiple of 8)
COLS	Maximum width of input and output image (Must be multiple of 8)
NPC	Number of pixels to be processed per cycle. This function supports only XF_NPPC1 or 1-pixel per cycle operations.
list_fix	A list of packed fixed point coordinates of the corner locations in 16, 5 (16 integer bits and 5 fractional bits) format. Bits from 20 to 0 represent the column number, while the bits 41 to 21 represent the row number. The rest of the bits are used for flag, this flag is set when the tracked corner is valid.
list	A list of packed positive short integer coordinates of the corner locations in unsigned short format. Bits from 15 to 0 represent the column number, while the bits 31 to 16 represent the row number. This list is same as the list output by Harris Corner Detector.
nCorners	Number of corners to track
flow_vectors	Packed flow vectors as in xf::DensePyrOpticalFlow function
harris_flag	If set to 1, the function takes input corners from list. if set to 0, the function takes input corners from list_fix.

The example codeworks on an input video which is read and processed using the xfOpenCV library. The core processing and tracking is done by the `xf_corner_tracker_accel()` function at the host.

cornersImgToList()

API Syntax

```
template <unsigned int MAXCORNERSNO, unsigned int TYPE, unsigned int ROWS,
unsigned int COLS, unsigned int NPC>
void cornersImgToList(xf::Mat<TYPE,ROWS,COLS,NPC> &_src, unsigned int
list[MAXCORNERSNO], unsigned int *ncorners)
```

Parameter Descriptions

The following table describes the template and theKintex® UltraScale+™ function parameters.

Table 446: **CornerImgToList Function Parameter Descriptions**

Parameter	Description
_src	The output image of harris corner detector. The size of this xf::Mat object is the size of the input image to Harris corner detector. The value of each pixel is 255 if a corner is present in the location, 0 otherwise.
list	A 32 bit memory allocated, the size of MAXCORNERS, to store the corners detected by Harris Detector

Table 446: **CornerImgToList Function Parameter Descriptions** *(cont'd)*

Parameter	Description
ncorners	Total number of corners detected by Harris, that is, the number of corners in the list

cornerTracker()

The `xf_corner_tracker_accel()` function does the core procesing and tracking at the host.

API Syntax

```
void cornerTracker(xf::Mat<XF_32UC1,HEIGHT,WIDTH,XF_NPPC1> & flow,
xf::Mat<XF_32UC1,HEIGHT,WIDTH,XF_NPPC1> & flow_iter,
xf::Mat<XF_8UC1,HEIGHT,WIDTH,XF_NPPC1> mat_imagepyr1[NUM_LEVELS] ,
xf::Mat<XF_8UC1,HEIGHT,WIDTH,XF_NPPC1> mat_imagepyr2[NUM_LEVELS] ,
xf::Mat<XF_8UC1, HEIGHT, WIDTH, XF_NPPC1> &inHarris, xf::Mat<XF_8UC1,
HEIGHT, WIDTH, XF_NPPC1> &outHarris, unsigned int *list, ap_uint<64>
*listfixed, int pyr_h[NUM_LEVELS], int pyr_w[NUM_LEVELS], unsigned int
*num_corners, unsigned int harrisThresh, bool *harris_flag)
```

Parameter Descriptions

The table below describes the template and the function parameters.

Parameter	Description
flow	Allocated xf::Mat to temporarily store the packed flow vectors during the iterative computation using the hardware function
flow_iter	Allocated xf::Mat to temporarily store the packed flow vectors during the iterative computation using the hardware function
mat_imagepyr1	An array, of size equal to the number of image pyramid levels, of xf::Mat to store the image pyramid of the first image
mat_imagepyr2	An array, of size equal to the number of image pyramid levels, of xf::Mat to store the image pyramid of the second image
inHarris	Input image to Harris Corner Detector in xf::Mat
outHarris	Output image from Harris detector. Image has 255 if a corner is present in the location and 0 otherwise
list	A 32 bit memory allocated, the size of MAXCORNERS, to store the corners detected by Harris Detector
listfixed	A 64 bit memory allocated, the size of MAXCORNERS, to store the corners tracked by xf::cornerUpdate
pyr_h	An array of integers the size of number of image pyramid levels to store the height of the image at each pyramid level
pyr_w	An array of integers the size of number of image pyramid levels to store the width of the image at each pyramid level
num_corners	An array, of size equal to the number ofNumber of corners detected by Harris Corner Detector
harrisThresh	Threshold input to the Harris Corner Detector, xf::harris
harris_flag	Flag used by the caller of this function to use the corners detected by xf::harris for the set of input images

Image Processing

The following steps demonstrate the Image Processing procedure in the hardware pipeline

1. `xf::cornerharris` is called to start processing the first input image
2. The output of `xf::cornerHarris` is pipelined by SDSoC™ on hardware to `xf::cornersImgToList`. This function takes in an image with corners marked as 255 and 0 elsewhere, and converts them to a list of corners.
3. Simultaneously, `xf::pyrDown` creates the two image pyramids and Dense Optical Flow is computed using the two image pyramids as described in the Iterative Pyramidal Dense Optical Flow example.
4. `xf::densePyrOpticalFlow` is called with the two image pyramids as inputs.
5. `xf::cornerUpdate` function is called to track the corner locations in the second image. If harris_flag is enabled, the `cornerUpdate` tracks corners from the output of the list, else it tracks the previously tracked corners.

```
if(*harris_flag == true)
    {
    #pragma SDS async(1)

xf::cornerHarris<FILTER_WIDTH,BLOCK_WIDTH,NMS_RADIUS,XF_8UC1,HEIGHT,WIDTH,XF
_NPPC1,XF_USE_URAM>(inHarris, outHarris, Thresh, k);
    #pragma SDS async(2)

xf::cornersImgToList<MAXCORNERS,XF_8UC1,HEIGHT,WIDTH,XF_NPPC1>(outHarris,
list, &nCorners);
    }
    //Code to compute Iterative Pyramidal Dense Optical Flow
    if(*harris_flag == true)
    {
    #pragma SDS wait(1)
    #pragma SDS wait(2)
        *num_corners = nCorners;
    }
    if(flag_flowin)
    {

xf::cornerUpdate<MAXCORNERS,XF_32UC1,HEIGHT,WIDTH,XF_NPPC1>(listfixed,
list, *num_corners, flow_iter, (ap_uint<1>)(*harris_flag));
    }


else


{


xf::cornerUpdate<MAXCORNERS,XF_32UC1,HEIGHT,WIDTH,XF_NPPC1>(listfixed,
list, *num_corners, flow, (ap_uint<1>)(*harris_flag));
    }
    if(*harris_flag == true)
    {
        *harris_flag = false;
    }
```

The `xf_corner_tracker_accel()` function takes a flag called harris_flag which is set during the first frame or when the corners need to be redetected. The `xf::cornerUpdate` function outputs the updated corners to the same memory location as the output corners list of `xf::cornerImgToList`. This means that when harris_flag is unset, the corners input to the `xf::cornerUpdate` are the corners tracked in the previous cycle, that is, the corners in the first frame of the current input frames.

After the Dense Optical Flow is computed, if harris_flag is set, the number of corners that `xf::cornerharris` has detected and `xf::cornersImgToList` has updated is copied to num_corners variable which is one of the outputs of the `xf_corner_tracker_accel()` function. The other being the tracked corners list, listfixed. If harris_flag is set, `xf::cornerUpdate` tracks the corners in 'list' memory location, otherwise it tracks the corners in 'listfixed' memory location.

Color Detection

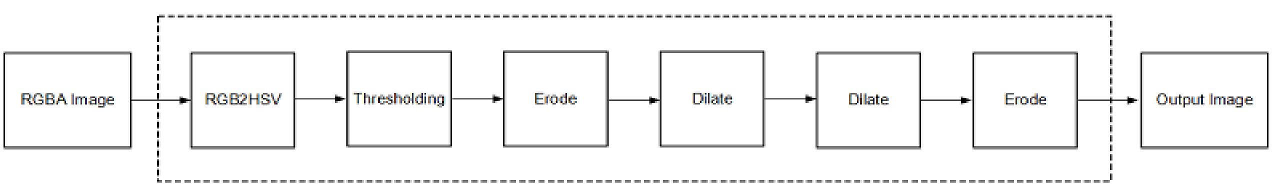
The Color Detection algorithm is basically used for color object tracking and object detection, based on the color of the object. The color based methods are very useful for object detection and segmentation, when the object and the background have a significant difference in color.

The Color Detection example uses four hardware functions from the xfOpenCV library. They are:

- xf::RGB2HSV
- xf::colorthresholding
- xf:: erode
- xf:: dilate

In the Color Detection example, the color space of the original BGR image is converted into an HSV color space. Because HSV color space is the most suitable color space for color based image segmentation. Later, based on the H (hue), S (saturation) and V (value) values, apply the thresholding operation on the HSV image and return either 255 or 0. After thresholding the image, apply erode (morphological opening) and dilate (morphological opening) functions to reduce unnecessary white patches (noise) in the image. Here, the example uses two hardware instances of erode and dilate functions. The erode followed by dilate and once again applying dilate followed by erode.

Figure 13: **Color Detection**



The following example demonstrates the Color Detection algorithm.

```
void colordetect_accel(xf::Mat<XF_8UC3, HEIGHT, WIDTH, XF_NPPC1> &_src,
        xf::Mat<XF_8UC3, HEIGHT, WIDTH, XF_NPPC1> &_rgb2hsv,
        xf::Mat<XF_8UC1, HEIGHT, WIDTH, XF_NPPC1> &_thresholdedimg,
        xf::Mat<XF_8UC1, HEIGHT, WIDTH, XF_NPPC1> &_erodeimage1,
        xf::Mat<XF_8UC1, HEIGHT, WIDTH, XF_NPPC1> &_dilateimage1,
        xf::Mat<XF_8UC1, HEIGHT, WIDTH, XF_NPPC1> &_dilateimage2,
        xf::Mat<XF_8UC1, HEIGHT, WIDTH, XF_NPPC1> &_dst,
        unsigned char *low_thresh, unsigned char *high_thresh){

xf::RGB2HSV< XF_8UC3,HEIGHT, WIDTH, XF_NPPC1>(_src, _rgb2hsv);
xf::colorthresholding<XF_8UC3,XF_8UC1,MAXCOLORS,HEIGHT,WIDTH,
XF_NPPC1>(_rgb2hsv,_  thresholdedimage, low_thresh, high_thresh);
xf::erode<XF_BORDER_CONSTANT,XF_8UC1,HEIGHT, WIDTH,
XF_NPPC1>(_thresholdedimg, _      erodeimage1);
    xf::dilate<XF_BORDER_CONSTANT,XF_8UC1,HEIGHT, WIDTH, XF_NPPC1>(_
erodeimage1, _ dilateimage1);
    xf::dilate<XF_BORDER_CONSTANT,XF_8UC1,HEIGHT, WIDTH, XF_NPPC1>(_
dilateimage1, _ dilateimage2);
    xf::erode<XF_BORDER_CONSTANT,XF_8UC1,HEIGHT, WIDTH, XF_NPPC1>(_
dilateimage2, _dst);

}
```

In the given example, the source image is passed to the `xf::RGB2HSV` function, the output of that function is passed to the `xf::colorthresholding` module, the thresholded image is passed to the `xf::erode` function and, the `xf::dilate` functions and the final output image are returned.

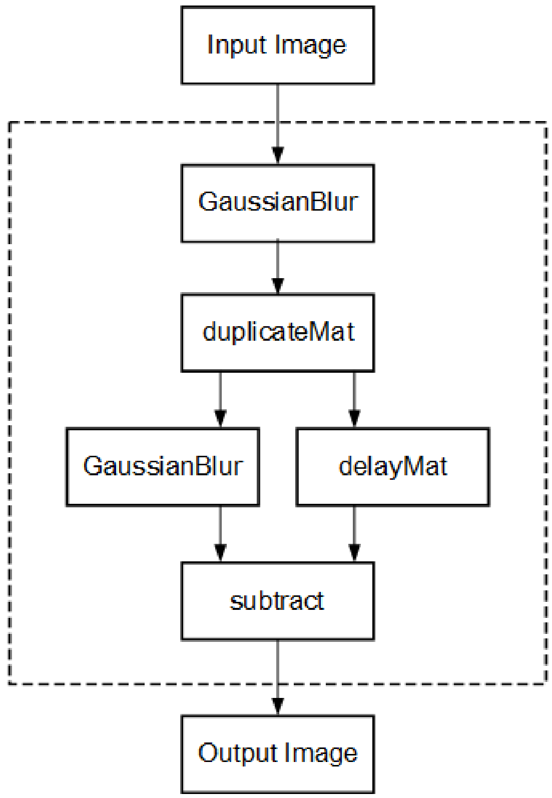
Difference of Gaussian Filter

The Difference of Gaussian Filter example uses four hardware functions from the xfOpenCV library. They are:

- xf::GaussianBlur
- xf::duplicateMat
- xf::delayMat
- xf::subtract

The Difference of Gaussian Filter function can be implemented by applying Gaussian Filter on the original source image, and that Gaussian blurred image is duplicated as two images. The Gaussian blur function is applied to one of the duplicated images, whereas the other one is stored as it is. Later, perform the Subtraction function on, two times Gaussian applied image and one of the duplicated image. Here, the duplicated image has to wait until the Gaussian applied for other one generates at least for one pixel output. Therefore, here xf::delayMat function is used to add delay.

Figure 14: **Difference of Gaussian Filter**



The following example demonstrates the Difference of Gaussian Filter example.

```
void gaussian_diff_accel(xf::Mat<XF_8UC1,HEIGHT,WIDTH,NPC1> &imgInput,
        xf::Mat<XF_8UC1,HEIGHT,WIDTH,XF_NPPC1> &imgin1,
        xf::Mat<XF_8UC1,HEIGHT,WIDTH, XF_NPPC1> &imgin2,
        xf::Mat<XF_8UC1,HEIGHT,WIDTH, XF_NPPC1> &imgin3,
        xf::Mat<XF_8UC1,HEIGHT,WIDTH, XF_NPPC1> &imgin4,
        xf::Mat<XF_8UC1,HEIGHT,WIDTH, XF_NPPC1> &imgin5,
        xf::Mat<XF_8UC1,HEIGHT,WIDTH, XF_NPPC1>&imgOutput,
float sigma)
{

    xf::GaussianBlur<FILTER_WIDTH, XF_BORDER_CONSTANT, XF_8UC1, HEIGHT,
WIDTH, XF_NPPC1>
(imgInput, imgin1, sigma);
    xf::duplicateMat<XF_8UC1, HEIGHT, WIDTH,
XF_NPPC1>(imgin1,imgin2,imgin3);
    xf::delayMat<MAXDELAY, XF_8UC1, HEIGHT, WIDTH, XF_NPPC1>(imgin3,imgin5);
    xf::GaussianBlur<FILTER_WIDTH, XF_BORDER_CONSTANT, XF_8UC1, HEIGHT,
WIDTH, XF_NPPC1>
(imgin2, imgin4, sigma);
xf::subtract<XF_CONVERT_POLICY_SATURATE, XF_8UC1, HEIGHT, WIDTH,
XF_NPPC1>(imgin5,imgin4,imgOutput);

}
```

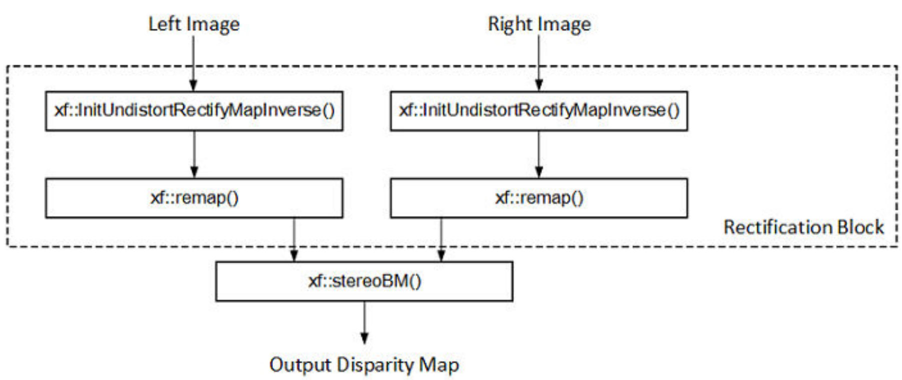
In the given example, the Gaussain Blur function is applied for source image `imginput`, and resultant image `imgin1` is passed to xf::duplicateMat. The `imgin2` and `imgin3` are the duplicate images of Gaussian applied image. Again gaussian blur is applied to `imgin2` and the result is stored in `imgin4`. Now, perform the subtraction between `imgin4` and `imgin3`, but here `imgin3` has to wait up to at least one pixel of `imgin4` generation. So, delay has applied for `imgin3` and stored in `imgin5`. Finally the subtraction performed on `imgin4` and `imgin5`.

Stereo Vision Pipeline

Disparity map generation is one of the first steps in creating a three dimensional map of the environment. The xfOpenCV library has components to build an image processing pipeline to compute a disparity map given the camera parameters and inputs from a stereo camera setup.

The two main components involved in the pipeline are stereo rectification and disparity estimation using local block matching method. While disparity estimation using local block matching is a discrete component in xfOpenCV, rectification block can be constructed using `xf::InitUndistortRectifyMapInverse()` and `xf::Remap()`. The dataflow pipeline is shown below. The camera parameters are an additional input to the pipeline.

Figure 15: **Stereo Vision Pipeline**



The following code is for the pipeline.

```
void stereopipeline_accel(xf::Mat<XF_8UC1, XF_HEIGHT, XF_WIDTH, XF_NPPC1>
&leftMat, xf::Mat<XF_8UC1, XF_HEIGHT, XF_WIDTH, XF_NPPC1> &rightMat,
xf::Mat<XF_16UC1, XF_HEIGHT, XF_WIDTH, XF_NPPC1> &dispMat,
    xf::Mat<XF_32FC1, XF_HEIGHT, XF_WIDTH, XF_NPPC1> &mapxLMat,
xf::Mat<XF_32FC1, XF_HEIGHT, XF_WIDTH, XF_NPPC1> &mapyLMat,
xf::Mat<XF_32FC1, XF_HEIGHT, XF_WIDTH, XF_NPPC1> &mapxRMat,
    xf::Mat<XF_32FC1, XF_HEIGHT, XF_WIDTH, XF_NPPC1> &mapyRMat,
xf::Mat<XF_8UC1, XF_HEIGHT, XF_WIDTH, XF_NPPC1> &leftRemappedMat,
xf::Mat<XF_8UC1, XF_HEIGHT, XF_WIDTH, XF_NPPC1> &rightRemappedMat,
    xf::xFSBMState<SAD_WINDOW_SIZE,NO_OF_DISPARITIES,PARALLEL_UNITS>
&bm_state, ap_fixed<32,12> *cameraMA_l_fix, ap_fixed<32,12>
*cameraMA_r_fix, ap_fixed<32,12> *distC_l_fix, ap_fixed<32,12>
```

```
*distC_r_fix,
    ap_fixed<32,12> *irA_l_fix, ap_fixed<32,12> *irA_r_fix, int _cm_size,
int _dc_size)
{

xf::InitUndistortRectifyMapInverse<XF_CAMERA_MATRIX_SIZE,XF_DIST_COEFF_SIZE,
XF_32FC1,XF_HEIGHT,XF_WIDTH,XF_NPPC1>(cameraMA_l_fix,distC_l_fix,irA_l_fix,m
apxLMat,mapyLMat,_cm_size,_dc_size);

xf::remap<XF_REMAP_BUFSIZE,XF_INTERPOLATION_BILINEAR,XF_8UC1,XF_32FC1,XF_8UC
1,XF_HEIGHT,XF_WIDTH,XF_NPPC1,XF_USE_URAM>(leftMat,leftRemappedMat,mapxLMat,
mapyLMat);


xf::InitUndistortRectifyMapInverse<XF_CAMERA_MATRIX_SIZE,XF_DIST_COEFF_SIZE,
XF_32FC1,XF_HEIGHT,XF_WIDTH,XF_NPPC1>(cameraMA_r_fix,distC_r_fix,irA_r_fix,m
apxRMat,mapyRMat,_cm_size,_dc_size);

xf::remap<XF_REMAP_BUFSIZE,XF_INTERPOLATION_BILINEAR,XF_8UC1,XF_32FC1,XF_8UC
1,XF_HEIGHT,XF_WIDTH,XF_NPPC1,XF_USE_URAM>(rightMat,rightRemappedMat,mapxRMa
t,mapyRMat);


xf::StereoBM<SAD_WINDOW_SIZE,NO_OF_DISPARITIES,PARALLEL_UNITS,XF_8UC1,XF_16U
C1,XF_HEIGHT,XF_WIDTH,XF_NPPC1,XF_USE_URAM>(leftRemappedMat,
rightRemappedMat, dispMat, bm_state);
}
```

Appendix A

Additional Resources and Legal Notices

Xilinx Resources

For support resources such as Answers, Documentation, Downloads, and Forums, see Xilinx Support.

Documentation Navigator and Design Hubs

Xilinx® Documentation Navigator (DocNav) provides access to Xilinx documents, videos, and support resources, which you can filter and search to find information. To open DocNav:

- From the Vivado® IDE, select **Help → Documentation and Tutorials**.
- On Windows, select **Start → All Programs → Xilinx Design Tools → DocNav**.
- At the Linux command prompt, enter `docnav`.

Xilinx Design Hubs provide links to documentation organized by design tasks and other topics, which you can use to learn key concepts and address frequently asked questions. To access the Design Hubs:

- In DocNav, click the **Design Hubs View** tab.
- On the Xilinx website, see the Design Hubs page.

Note: For more information on DocNav, see the Documentation Navigator page on the Xilinx website.

References

1. *SDSoC Environment Getting Started Tutorial* (UG1028)

2. *SDSoC Environment Tutorial: Platform Creation* (UG1236)
3. *UltraFast Embedded Design Methodology Guide* (UG1046)
4. *Zynq-7000 SoC Software Developers Guide* (UG821)
5. *Zynq UltraScale+ MPSoC: Software Developers Guide* (UG1137)
6. *ZC702 Evaluation Board for the Zynq-7000 XC7Z020 SoC User Guide* (UG850)
7. *ZCU102 Evaluation Board User Guide* (UG1182)
8. *PetaLinux Tools Documentation: Reference Guide* (UG1144)
9. *Vivado Design Suite User Guide: High-Level Synthesis* (UG902)
10. *Vivado Design Suite User Guide: Creating and Packaging Custom IP* (UG1118)
11. SDSoC Development Environment web page
12. Vivado® Design Suite Documentation

Please Read: Important Legal Notices

AUTOMOTIVE APPLICATIONS DISCLAIMER

AUTOMOTIVE PRODUCTS (IDENTIFIED AS "XA" IN THE PART NUMBER) ARE NOT WARRANTED FOR USE IN THE DEPLOYMENT OF AIRBAGS OR FOR USE IN APPLICATIONS THAT AFFECT CONTROL OF A VEHICLE ("SAFETY APPLICATION") UNLESS THERE IS A SAFETY CONCEPT OR REDUNDANCY FEATURE CONSISTENT WITH THE ISO 26262 AUTOMOTIVE SAFETY STANDARD ("SAFETY DESIGN"). CUSTOMER SHALL, PRIOR TO USING OR DISTRIBUTING ANY SYSTEMS THAT INCORPORATE PRODUCTS, THOROUGHLY TEST SUCH SYSTEMS FOR SAFETY PURPOSES. USE OF PRODUCTS IN A SAFETY APPLICATION WITHOUT A SAFETY DESIGN IS FULLY AT THE RISK OF CUSTOMER, SUBJECT ONLY TO APPLICABLE LAWS AND REGULATIONS GOVERNING LIMITATIONS ON PRODUCT LIABILITY.

Copyright