

# ザイリンクス OpenCV ユーザーガイド

UG1233 (v2019.1) 2019 年 6 月 5 日

この資料は表記のバージョンの英語版を翻訳したもので、内容に相違が生じる場合には原文を優先します。資料によっては英語版の更新に対応していないものがあります。日本語版は参考用としてご使用の上、最新情報につきましては、必ず最新英語版をご参照ください。

# 改訂履歴

次の表に、この文書の改訂履歴を示します。

セクション	改訂内容
2019 年 6 月 5 日 バージョン 2019.1	
資料全体	2019.1 リリース用にマイナーな編集上の更新
<a href="#">第 4 章: HLS の使用</a>	新しいセクションを追加
<a href="#">第 3 章: SDAccel での使用方法</a>	存在の機能を格納
<a href="#">カルマン フィルター</a>	拡張カルマン フィルターのサポートを追加
<a href="#">色変換</a>	色変換フォーマットを追加
<a href="#">境界ボックス (BoundingBox)</a>	新しい関数を追加
<a href="#">切り抜き (crop)</a>	新しい関数を追加
<a href="#">xfOpenCV ライブラリ関数</a>	いくつかの関数に色画像サポートを追加
<a href="#">xf::Mat 画像コンテナー クラス</a>	マイナー アップデート
<a href="#">ワープ変換 (warpTransform)</a>	warpAffine および warpPerspective のサポートを追加

# 目次

改訂履歴.....	2
第 1 章: 概要.....	5
基本的な機能.....	5
reVISION プラットフォーム上の xfOpenCV カーネル.....	6
xfOpenCV ライブラリの内容.....	8
第 2 章: SDSoC での使用方法.....	9
使用要件.....	9
HLS ビデオ ライブラリの xfOpenCV への移行.....	10
xfOpenCV ライブラリの使用.....	20
ハードウェア カーネル コンフィギュレーションの変更.....	32
xfOpenCV ライブラリ関数のハードウェアでの使用.....	33
第 3 章: SDAccel での使用方法.....	37
使用要件.....	37
SDAccel 設計手法.....	37
機能の評価.....	44
第 4 章: HLS の使用.....	46
HLS スタンドアロン モード.....	46
AXI ビデオ インターフェイス関数.....	48
第 5 章: xfOpenCV ライブラリ API リファレンス.....	52
xf::Mat 画像コンテナ クラス.....	52
xfOpenCV ライブラリ関数.....	61
第 6 章: xfOpenCV ライブラリを使用したデザイン例.....	259
反復ピラミッド型高密度オプティカル フロー.....	259
疎なオプティカル フローを使用したコーナー追跡.....	263
色検出.....	268
ガウシアン フィルターの差.....	269
ステレオ ビジョン パイプライン.....	270
付録 A: その他のリソースおよび法的通知.....	272
ザイリンクス リソース.....	272
Documentation Navigator およびデザイン ハブ.....	272
参考資料.....	272

お読みください: 重要な法的通知.....	273
-----------------------	-----

## 概要

この資料では、xfOpenCV ライブラリを FPGA デバイス用に最適化したザイリンクス xfOpenCV ライブラリについて説明します。Zynq<sup>®</sup>-7000 SoC、Zynq<sup>®</sup> UltraScale+<sup>™</sup> MPSoC、および PCIe ベース デバイス (Virtex、U200 など) を使用するアプリケーション開発者向けに記述されています。xfOpenCV ライブラリは、SDx<sup>™</sup> 開発環境で使用するために設計されており、FPGA デバイス上でアクセラレーションされるコンピュータ ビジョン関数用のソフトウェアインターフェイスを提供します。xfOpenCV ライブラリ関数の機能は、OpenCV の等価関数とほぼ同じです。違いがある場合は、このユーザー ガイドに記述されています。

**注記:** xfOpenCV ライブラリの使用要件は、[使用要件](#)を参照してください。xfOpenCV ライブラリ関数を使用するための手順の詳細は、[xfOpenCV ライブラリの使用](#)を参照してください。

---

## 基本的な機能

xfOpenCV ライブラリ関数は、すべて一般的なフォーマットに従っています。すべての関数に次の特徴があります。

- すべての関数はテンプレートとして設計されており、画像である引数はすべて `xf::Mat` として供給する必要があります。
- すべての関数は `xf` 名前空間で定義されています。
- 主なテンプレート引数は次のとおりです。
  - 処理する画像の最大サイズ
  - 各ピクセルのプロパティを定義するデータ型
  - 1 クロック サイクルごとに処理されるピクセル数
  - 機能に関連するその他のコンパイル時引数

xfOpenCV ライブラリには列挙データ型が含まれており、ユーザーが `xf::Mat` を設定できます。`xf::Mat` の詳細は、[xf::Mat 画像コンテナ クラス](#)を参照してください。

## reVISION プラットフォーム上の xfOpenCV カーネル

xfOpenCV ライブラリは、SDx 開発環境でできるように設計されています。xfOpenCV カーネルは、reVISION プラットフォームで評価されています。

入力と出力の両方が画像ファイルであるサンプル デザインの一般的なフローは、次のとおりです。

1. `cv::imread()` を使用して画像を読み込みます。
2. データを `xf::Mat` にコピーします。
3. xfOpenCV で処理関数を呼び出します。
4. データを `xf::Mat` から `cv::Mat` にコピーします。
5. `cv::imwrite()` を使用して出力を画像に書き出します。

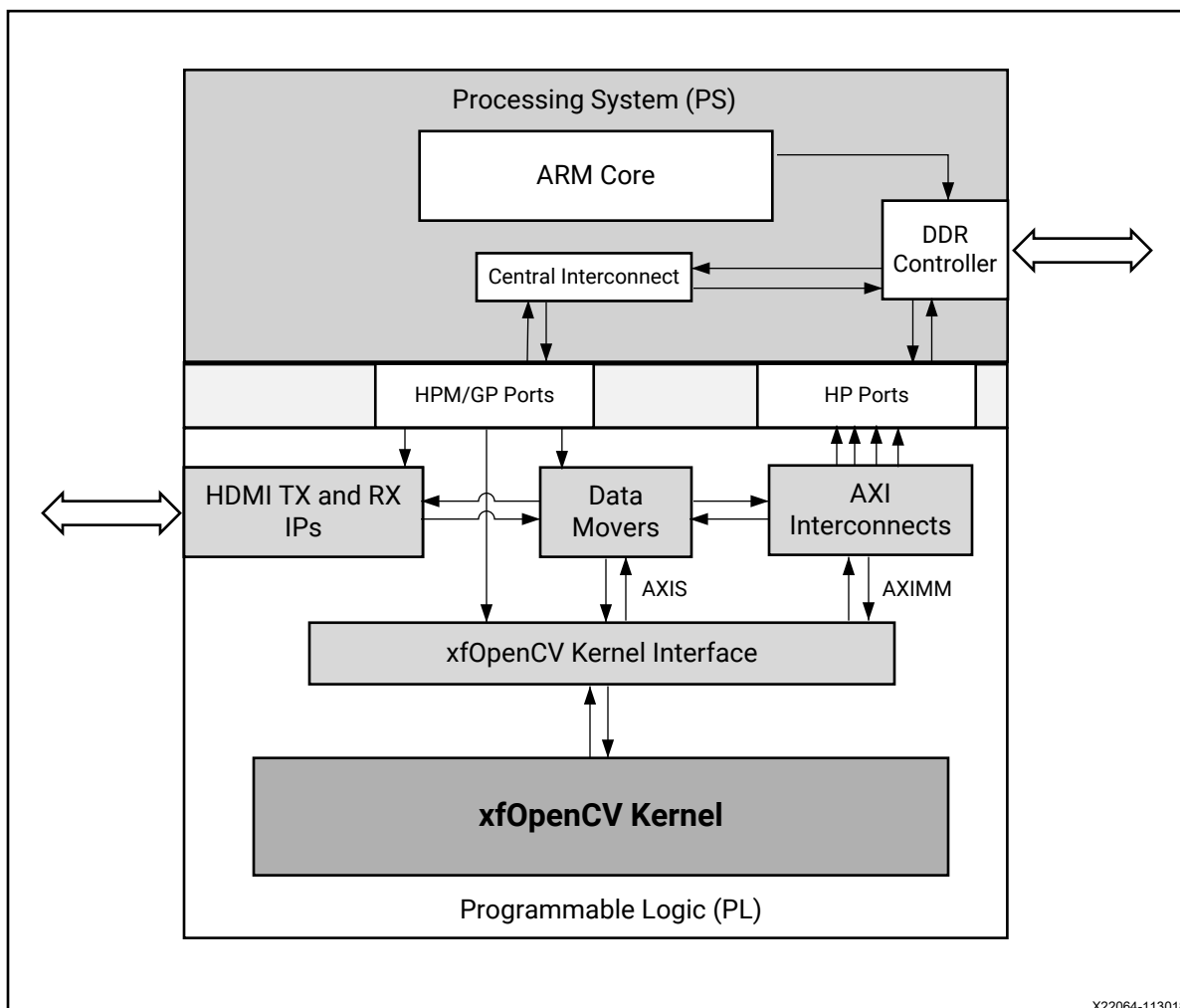
コード全体がパイプラインのホスト コードとして書き出され、そこから xfOpenCV 関数へのすべての呼び出しがハードウェアに移動されます。xfOpenCV からの関数は、メモリ内の画像の読み出しおよび書き込みに使用されます。xfOpenCV ライブラリ関数の画像コンテナは、`xf::Mat` オブジェクトです。詳細は、[xf::Mat 画像コンテナ クラス](#)を参照してください。

reVISION プラットフォームでは、ライブおよびファイル入力/出力 (I/O) モードがサポートされます。詳細は、[「reVISION 入門ガイド」](#)を参照してください。

- ファイル I/O モードでは、コントローラーで画像が SD カードからハードウェア カーネルに転送されます。ファイル I/O モードの手順は、次のとおりです。
  1. プロセッシング システム (PS) で SD カードから画像フレームを読み出し、DRAM に格納します。
  2. xfOpenCV カーネルで DRAM から画像を読み出して処理し、出力を DRAM メモリに戻します。
  3. PS で DRAM から出力画像フレームを読み出し、SD カードに戻します。
- ライブ I/O モードでは、フレームをプラットフォームにストリーミングし、xfOpenCV カーネルでフレームを処理して、適切なインターフェイスを介してフレームをストリーミング出力します。ライブ I/O モードの手順は、次のとおりです。
  1. ビデオ キャプチャ IP でフレームを受信し、DRAM に格納します。
  2. xfOpenCV カーネルで DRAM から画像をフェッチして処理し、出力を DRAM に格納します。
  3. 表示 IP で DRAM から出力フレームを読み出し、画像インターフェイスを介してフレームを出力します。

次の図に、xfOpenCV ブロックを含む reVISION プラットフォームを示します。

図 1: reVISION プラットフォーム上の xfOpenCV カーネル



**注記:** PS-PL インターフェイスおよび PL-DDR インターフェイスの詳細は、『Zynq UltraScale+ デバイス テクニカル リファレンス マニュアル』 (UG1085: [英語版](#)、[日本語版](#)) を参照してください。

## xfOpenCV ライブラリの内容

次の表に、xfOpenCV ライブラリの内容を示します。

表 1: xfOpenCV ライブラリの内容

フォルダー	詳細
include	ライブラリに必要なヘッダー ファイルが含まれます。
include/common	ライブラリ特有のタイプなど、共通するライブラリの構造ヘッダーが含まれます。
include/core	math 関数などのコア ライブラリの機能ヘッダーが含まれます。
include/features	特徴抽出カーネル関数定義が含まれます (例: Harris)。
include/imgproc	features フォルダーで提供されるものを除くすべてのカーネル関数定義が含まれます。
include/video	features and imgproc フォルダーで提供されるものを除くすべてのカーネル関数定義が含まれます。
examples	ユニット テストを実行するサンプルテストベンチ コードが含まれます。examples/ フォルダーには、アルゴリズム名のフォルダーが含まれます。各アルゴリズム フォルダーには、ホストファイル、.json ファイル、および data フォルダーが含まれます。xfOpenCV ライブラリの使用の詳細は、 <a href="#">reVISION プラットフォーム上の xfOpenCV カーネル</a> を詳細してください。
examples	SDAccel™ 環境で xfOpenCV ライブラリを使用する方法を示す 24 個の関数のサンプルテストベンチ コードが含まれます。
HLS_Use_Model	スタンドアロン Vivado HLS で 2 つの異なるモードで xfOpenCV 関数を使用する例が含まれます。
HLS_Use_Model/Standalone_HLS_Example	スタンドアロン Vivado HLS ツールで xfOpenCV 関数をそのまま合成するためのサンプル コードと Tcl スクリプトが含まれます。
HLS_Use_Model/Standalone_HLS_AXI_Example	スタンドアロン Vivado HLS ツールで AXI インターフェイスを使用する関数を合成するためのサンプル コードと Tcl スクリプトが含まれます。



# SDSoC での使用方法

この章では、xfOpenCV ライブラリ関数を使用してデザインを作成するために必要な情報を示します。

## 使用要件

このセクションでは、ZCU104 ベースのプラットフォームで xfOpenCV ライブラリ関数を使用するための要件を示します。ZC702 および ZC706 reVISION プラットフォームにも同じ要件が適用されます。

- 『SDSoC 環境リリース ノート、インストール、およびライセンス ガイド』 ([UG1294](#)) の手順に従って SDx 開発環境をダウンロードおよびインストールします。プロジェクトのビルドにターミナルを使用する場合は、SDx 開発環境を起動する前に、\$SYSROOT 環境変数を reVISION プラットフォームで提供される Linux ルート ファイル システムに設定してください。次に例を示します。

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

- Zynq® UltraScale+™ MPSoC エンベデッド ビジョン プラットフォーム ZIP ファイルをダウンロードして解凍します。解凍したデザイン ファイル階層の zcu104\_rv\_ss フォルダに SDx 開発環境ワークスペースを作成します。詳細は、「[reVISION 入門ガイド](#)」を参照してください。
- ZCU104 評価ボードを設定します。詳細は、「[reVISION 入門ガイド](#)」を参照してください。
- xfOpenCV ライブラリをダウンロードします。このライブラリは GitHub から入手できます。次の `git clone` コマンドを実行し、xfOpenCV リポジトリをローカル ディスクにクローンします。

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

## HLS ビデオ ライブラリの xfOpenCV への移行

HLS ビデオ ライブラリは、廃止予定です。HLS ビデオ ライブラリで使用可能なすべての関数とほとんどのインフラストラクチャは、xfOpenCV で提供されますが、名前および機能の一部が変更されています。xfOpenCV に移植されたこれらの HLS ビデオ ライブラリ関数では、SDSoc ビルド フローもサポートされます。

このセクションでは、HLS ビデオ ライブラリに含まれる C++ ビデオ処理関数およびインフラストラクチャを使用している場合の詳細を示します。

### インフラストラクチャ関数およびクラス

HLS ビデオ ライブラリからインポートされたすべての関数では、xfOpenCV ライブラリと同様、画像データを表すのに hls::Mat ではなく xf::Mat が使用されます。これらの主な違いは、hls::Mat ではデータの格納に hls::stream が使用されるのに対し、xf::Mat ではポインターが使用されることです。そのため、移行の際に hls::Mat を厳密に xf::Mat に置き換えることはできません。

次の表に、hls::Mat と xf::Mat のメンバー関数の違いを示します。

表 2: インフラストラクチャ関数およびクラス

メンバー関数	hls::Mat (HLS ビデオ ライブラリ)	xf::Mat (xfOpenCV ライブラリ)
channels()	チャンネル数を返す	チャンネル数を返す
type()	ピクセルのデータ型の列挙値を返す	ピクセルのデータ型の列挙値を返す
depth()	ピクセルのデータ型の列挙値を返す	チャンネルを含むピクセルの深さを返す
read()	ストリームから値を読み出し、スカラー値として返す	指定の場所から値を読み出し、複数ピクセル/クロックのパケット値として返します。
operator >>	read() と同様	xfOpenCV にはなし
operator <<	write() と同様	xfOpenCV にはなし
Write()	スカラー値のをストリームに書き込み	複数ピクセル/クロックのパケット値を指定の場所に書き込みます。

HLS ビデオ ライブラリに含まれるインフラストラクチャ ファイル hls\_video\_core.h、hls\_video\_mem.h、hls\_video\_types.h は、xfOpenCV ライブラリの xf\_video\_core.h、xf\_video\_mem.h、xf\_video\_types.h に移行されており、hls\_video\_imgbase.h は廃止されています。これらのファイルに含まれるコードは、xf::namespace にあることを除き、変更されていません。

### クラス

- メモリ ウィンドウ バッファ: hls::window は xf::window に移行されています。名前空間以外は、インプリメンテーションは変更されていません。これは、xf\_video\_mem.h ファイルに含まれます。
- メモリ ライン バッファ: hls::LineBuffer は xf::LineBuffer に移行されています。xf::LineBuffer にストレージ構造に異なるタイプの RAM 構造を推論するテンプレート引数が追加されている以外は、これら 2 つに違いはありません。デフォルトのストレージ タイプは RAM\_S2P\_BRAM で、RESHAPE\_FACTOR=1 に設定されています。詳細は、[xf::LineBuffer](#) を参照してください。これは、xf\_video\_mem.h ファイルに含まれます。

## 関数

- OpenCV インターフェイス関数: これらの関数は、画像データを OpenCV Mat フォーマットと HLS AXI タイプの間で変換します。HLS ビデオ ライブラリには 14 個のインターフェイス関数が含まれており、そのうち 2 つ (cvMat2AXIvideo および AXIvideo2cvMat) が xfOpenCV ライブラリの xf\_axi.h に含まれます。それ以外の関数はすべて廃止されています。
- AXI4-Stream I/O 関数: hls::Mat と AXI4-Stream 準拠のデータ型 (hls::stream) の間の変換を実行する I/O 関数は、hls::AXIvideo2Mat および hls::Mat2AXIvideo です。これらの関数は廃止されており、xf::Mat との変換用に 2 つの新しい関数 xf::AXIvideo2xfMat および xf::xfMat2AXIvideo が追加されています。これらの関数を使用するには、ヘッダー ファイル xf\_infra.h を含める必要があります。

## xf::window

2D ウィンドウ バッファーを表現するテンプレート クラス。ウィンドウ バッファーの行数、列数、およびピクセルのデータ型を指定する 3 つのパラメーターがあります。

### クラス定義

```
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
};
```

### パラメーターの説明

次の表に、xf::window クラスのメンバーとその説明を示します。

表 3: windos 関数のパラメーターの説明

パラメーター	説明
Val	バッファーの内容を保持する 2-D 配列。

## メンバー関数の説明

表 4: メンバー関数の説明

関数	説明
shift_pixels_left()	ウィンドウを左にシフトします。ウィンドウ内に格納されているデータはすべて右に移動し、一番左の列 (COLS-1) は空になり、新しいデータを挿入できるようになります。
shift_pixels_right()	ウィンドウを右にシフトします。ウィンドウ内に格納されているデータはすべて左側に移動し、一番右の列 (0) は空になり、新しいデータを挿入できるようになります。
shift_pixels_up()	ウィンドウを上をシフトします。ウィンドウ内に格納されているデータはすべて下に移動し、一番上の行 (ROWS-1) は空になり、新しいデータを挿入できるようになります。
shift_pixels_down()	ウィンドウを下にシフトします。ウィンドウ内に格納されているデータはすべて上に移動し、一番下の行 (0) は空になり、新しいデータを挿入できるようになります。
insert_pixel(T value, int row, int col)	新しい要素値をウィンドウの指定の位置 (row, column) に挿入します。
insert_row(T value[COLS], int row)	値のセットをウィンドウの任意の行に挿入します。
insert_top_row(T value[COLS])	値のセットをウィンドウの一番上の行 (0) に挿入します。
insert_bottom_row(T value[COLS])	値のセットをウィンドウの一番下の行 (ROWS-1) に挿入します。
insert_col(T value[ROWS], int col)	値のセットをウィンドウの任意の列に挿入します。
insert_left_col(T value[ROWS])	値のセットをウィンドウの一番左の列 (0) に挿入します。
insert_right_col(T value[ROWS])	値のセットをウィンドウの一番右の列 (COLS-1) に挿入します。
T& getval(int row, int col)	ウィンドウの指定した位置 (row, column) のデータ値を返します。
T& operator()(int row, int col)	ウィンドウの指定した位置 (row, column) のデータ値を返します。
restore_val()	ウィンドウ バッファの内容を別の配列に復元します。
window_print()	ウィンドウ バッファに含まれるすべてのデータをコンソールに表示します。

## テンプレートのパラメーターの説明

表 5: テンプレートのパラメーターの説明

パラメーター	説明
ROWS	ウィンドウ バッファの行数。
COLS	ウィンドウ バッファの列数。
T	ウィンドウ バッファのピクセルのデータ型。

## ウィンドウ バッファ宣言のサンプル コード

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

## xf::LineBuffer

2D ラインバッファを表現するテンプレートクラス。ウィンドウバッファの行数、列数、およびピクセルのデータ型を指定する 3 つのパラメーターがあります。

### クラス定義

```
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& get_val(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
};
```

### パラメーターの説明

次の表に、xf::LineBuffer クラスのメンバーとその説明を示します。

表 6: ライン バッファ関数のパラメーターの説明

パラメーター	説明
Val	ライン バッファの内容を保持する 2-D 配列。

### メンバー関数の説明

表 7: メンバー関数の説明

関数	説明
shift_pixels_up(int col)	ライン バッファの内容を上シフトします。新しい値が一番下の行 (ROWS-1) に配置されます。
shift_pixels_down(int col)	ライン バッファの内容を下シフトします。新しい値が一番上の行 (0) に配置されます。
insert_bottom_row(T value, int col)	新しい値をライン バッファの一番下の行 (ROWS-1) に挿入します。

表 7: メンバー関数の説明 (続き)

関数	説明
insert_top_row(T value, int col)	新しい値をラインバッファの一番上の行 (0) に挿入します。
get_col(T value[ROWS], int col)	ラインバッファの列の値を取得します。
T& getval(int row, int col)	ラインバッファの指定した位置 (row, column) のデータ値を返します。
T& operator()(int row, int col);	ラインバッファの指定した位置 (row, column) のデータ値を返します。

## テンプレートのパラメーターの説明

表 8: テンプレートのパラメーターの説明

パラメーター	説明
ROWS	ラインバッファの行数。
COLS	ラインバッファの列数。
T	ラインバッファのピクセルのデータ型。
MEM_TYPE	ストレージ要素のタイプ。有効な値は、RAM_1P_BRAM、RAM_1P_URAM、RAM_2P_BRAM、RAM_2P_URAM、RAM_S2P_BRAM、RAM_S2P_URAM、RAM_T2P_BRAM、RAM_T2P_URAM です。
RESHAPE_FACTOR	配列を分割する量を指定します。

次に、ラインバッファ宣言のサンプルコードを示します。

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

## ビデオ処理関数

次の表に、HLS ビデオ ライブラリから xfOpenCV ライブラリに移植されたビデオ処理関数を API の変更点と共に示します。

表 9: ビデオ処理関数

関数	HLS ビデオ ライブラリ API	xfOpenCV ライブラリ API
addS	<pre>template&lt;int ROWS, int COLS, int SRC_T, typename T, int DST_T&gt; void AddS(Mat&lt;ROWS, COLS, SRC_T&gt;&amp;src, Scalar&lt;HLS_MAT_CN(SRC_T), _T&gt;_scl, Mat&lt;ROWS, COLS, DST_T&gt;&amp;_dst)</pre>	<pre>template&lt;int POLICY_TYPE, int SRC_T, int ROWS, int COLS, int NPC = 1&gt; void addS(xf::Mat&lt;SRC_T, ROWS, COLS, NPC&gt; &amp;_src1, unsigned char _scl[XF_CHANNELS(SRC_T, NPC)], xf::Mat&lt;SRC _T, ROWS, COLS, NPC&gt; &amp;_dst)</pre>
AddWeighted	<pre>template&lt;int ROWS, int COLS, int SRC1_T, int SRC2_T, int DST_T, typename P_T&gt; void AddWeighted(Mat&lt;ROWS, COLS, SRC1_T&gt;&amp; src1, P_T alpha, Mat&lt;ROWS, COLS, SRC2_T&gt;&amp; src2, P_T beta, P_T gamma, Mat&lt;ROWS, COLS, DST_T&gt;&amp; dst)</pre>	<pre>template&lt; int SRC_T, int DST_T, int ROWS, int COLS, int NPC = 1&gt; void addWeighted(xf::Mat&lt;SRC_T, ROWS, COLS, NPC&gt; &amp; src1, float alpha, xf::Mat&lt;SRC_T, ROWS, COLS, NPC&gt; &amp; src2, float beta, float gama, xf::Mat&lt;DST_T, ROWS, COLS, NPC&gt; &amp; dst)</pre>
Cmp	<pre>template&lt;int ROWS, int COLS, int SRC1_T, int SRC2_T, int DST_T&gt; void Cmp(Mat&lt;ROWS, COLS, SRC1_T&gt;&amp; src1, Mat&lt;ROWS, COLS, SRC2_T&gt;&amp; src2, Mat&lt;ROWS, COLS, DST_T&gt;&amp; dst, int cmp_op)</pre>	<pre>template&lt;int CMP_OP, int SRC_T, int ROWS, int COLS, int NPC = 1&gt; void compare(xf::Mat&lt;SRC_T, ROWS, COLS, NPC&gt; &amp;_src1, xf::Mat&lt;SRC_T, ROWS, COLS, NPC&gt; &amp;_src2, xf::Mat&lt;SRC_T, ROWS, COLS, NPC&gt; &amp;_dst)</pre>
CmpS	<pre>template&lt;int ROWS, int COLS, int SRC_T, typename P_T, int DST_T&gt; void CmpS(Mat&lt;ROWS, COLS, SRC_T&gt;&amp; src, P_T value, Mat&lt;ROWS, COLS, DST_T&gt;&amp; dst, int cmp_op)</pre>	<pre>template&lt;int CMP_OP, int SRC_T, int ROWS, int COLS, int NPC = 1&gt; void compare(xf::Mat&lt;SRC_T, ROWS, COLS, NPC&gt; &amp;_src1, unsigned char _scl[XF_CHANNELS(SRC_T, NPC)], xf::Mat&lt;SRC _T, ROWS, COLS, NPC&gt; &amp;_dst)</pre>
Max	<pre>template&lt;int ROWS, int COLS, int SRC1_T, int SRC2_T, int DST_T&gt; void Max(Mat&lt;ROWS, COLS, SRC1_T&gt;&amp; src1, Mat&lt;ROWS, COLS, SRC2_T&gt;&amp; src2, Mat&lt;ROWS, COLS, DST_T&gt;&amp; dst)</pre>	<pre>template&lt;int SRC_T, int ROWS, int COLS, int NPC = 1&gt; void Max(xf::Mat&lt;SRC_T, ROWS, COLS, NPC&gt; &amp;_src1, xf::Mat&lt;SRC_T, ROWS, COLS, NPC&gt; &amp;_src2, xf::Mat&lt;SRC_T, ROWS, COLS, NPC&gt; &amp;_dst)</pre>
MaxS	<pre>template&lt;int ROWS, int COLS, int SRC_T, typename T, int DST_T&gt; void MaxS(Mat&lt;ROWS, COLS, SRC_T&gt;&amp; src, _T value, Mat&lt;ROWS, COLS, DST_T&gt;&amp; dst)</pre>	<pre>template&lt; int SRC_T, int ROWS, int COLS, int NPC = 1&gt; void max(xf::Mat&lt;SRC_T, ROWS, COLS, NPC&gt; &amp;_src1, unsigned char _scl[XF_CHANNELS(SRC_T, NPC)], xf::Mat&lt;SRC _T, ROWS, COLS, NPC&gt; &amp;_dst)</pre>
Min	<pre>template&lt;int ROWS, int COLS, int SRC1_T, int SRC2_T, int DST_T&gt; void Min(Mat&lt;ROWS, COLS, SRC1_T&gt;&amp; src1, Mat&lt;ROWS, COLS, SRC2_T&gt;&amp; src2, Mat&lt;ROWS, COLS, DST_T&gt;&amp; dst)</pre>	<pre>template&lt; int SRC_T, int ROWS, int COLS, int NPC = 1&gt; void Min(xf::Mat&lt;SRC_T, ROWS, COLS, NPC&gt; &amp;_src1, xf::Mat&lt;SRC_T, ROWS, COLS, NPC&gt; &amp;_src2, xf::Mat&lt;SRC_T, ROWS, COLS, NPC&gt; &amp;_dst)</pre>

表 9: ビデオ処理関数 (続き)

関数	HLS ビデオ ライブラリ API	xfOpenCV ライブラリ API
MinS	<pre>template&lt;int ROWS, int COLS, int SRC_T, typename T, int DST_T&gt; void MinS(Mat&lt;ROWS, COLS, SRC_T&gt;&amp; src,           T value, Mat&lt;ROWS, COLS, DST_T&gt;&amp; dst)</pre>	<pre>template&lt; int SRC_T, int ROWS, int COLS, int NPC =1&gt; void min(xf::Mat&lt;SRC_T, ROWS, COLS, NPC&gt; &amp; _src1, unsigned char _scl[XF_CHANNELS(SRC_T,NPC)],xf::Mat&lt;SRC _T, ROWS, COLS, NPC&gt; &amp; _dst)</pre>
PaintMask	<pre>template&lt;int SRC_T,int MASK_T,int ROWS,int COLS&gt; void PaintMask( Mat&lt;ROWS,COLS,SRC_T&gt; &amp;_src, Mat&lt;ROWS,COLS,MASK_T&gt;&amp; mask, Mat&lt;ROWS,COLS,SRC_T&gt;&amp; _dst,Scalar&lt;HLS_MAT _CN(SRC_T),HLS_TNAME(SRC_T)&gt; _color)</pre>	<pre>template&lt; int SRC_T,int MASK_T, int ROWS, int COLS,int NPC=1&gt; void paintmask(xf::Mat&lt;SRC_T, ROWS, COLS, NPC&gt; &amp; _src_mat, xf::Mat&lt;MASK_T, ROWS, COLS, NPC&gt; &amp; in_mask, xf::Mat&lt;SRC_T, ROWS, COLS, NPC&gt; &amp; _dst_mat, unsigned char _color[XF_CHANNELS(SRC_T,NPC)])</pre>
Reduce	<pre>template&lt;typename INTER_SUM T, int ROWS, int COLS, int SRC_T, int DST_ROWS, int DST_COLS, int DST_T&gt; void Reduce( Mat&lt;ROWS, COLS, SRC_T&gt; &amp;src, Mat&lt;DST_ROWS, DST_COLS, DST_T&gt; &amp;dst, int dim, int op=HLS_REDUCE_SUM)</pre>	<pre>template&lt; int REDUCE_OP, int SRC_T,int DST_T, int ROWS, int COLS,int ONE_D_HEIGHT, int ONE_D_WIDTH, int NPC=1&gt; void reduce(xf::Mat&lt;SRC_T, ROWS, COLS, NPC&gt; &amp; _src_mat, xf::Mat&lt;DST_T, ONE_D_HEIGHT, ONE_D_WIDTH, 1&gt; &amp; _dst_mat, unsigned char dim)</pre>
zero	<pre>template&lt;int ROWS, int COLS, int SRC_T, int DST_T&gt; void Zero(Mat&lt;ROWS, COLS, SRC_T&gt;&amp; src, Mat&lt;ROWS, COLS, DST_T&gt;&amp; dst)</pre>	<pre>template&lt; int SRC_T, int ROWS, int COLS, int NPC =1&gt; void zero(xf::Mat&lt;SRC_T, ROWS, COLS, NPC&gt; &amp; _src1,xf::Mat&lt;SRC_T, ROWS, COLS, NPC&gt; &amp; _dst)</pre>
Sum	<pre>template&lt;typename DST_T, int ROWS, int COLS, int SRC_T&gt; Scalar&lt;HLS_MAT_CN(SRC_T), DST_T&gt; Sum( Mat&lt;ROWS, COLS, SRC_T&gt;&amp; src)</pre>	<pre>template&lt; int SRC_T, int ROWS, int COLS, int NPC = 1&gt; void sum(xf::Mat&lt;SRC_T, ROWS, COLS, NPC&gt; &amp; _src1, double sum[XF_CHANNELS(SRC_T,NPC)] )</pre>
SubS	<pre>template&lt;int ROWS, int COLS, int SRC_T, typename T, int DST_T&gt; void SubS(Mat&lt;ROWS, COLS, SRC_T&gt;&amp; src, Scalar&lt;HLS_MAT_CN(SRC_T), T&gt; scl, Mat&lt;ROWS, COLS, DST_T&gt;&amp; dst)</pre>	<pre>template&lt;int POLICY_TYPE, int SRC_T, int ROWS, int COLS, int NPC =1&gt; void SubS(xf::Mat&lt;SRC_T, ROWS, COLS, NPC&gt; &amp; _src1, unsigned char _scl[XF_CHANNELS(SRC_T,NPC)],xf::Mat&lt;SRC _T, ROWS, COLS, NPC&gt; &amp; _dst)</pre>
subRS	<pre>template&lt;int ROWS, int COLS, int SRC_T, typename T, int DST_T&gt; void SubRS(Mat&lt;ROWS, COLS, SRC_T&gt;&amp; src, Scalar&lt;HLS_MAT_CN(SRC_T), T&gt; scl, Mat&lt;ROWS, COLS, DST_T&gt;&amp; dst)</pre>	<pre>template&lt;int POLICY_TYPE, int SRC_T, int ROWS, int COLS, int NPC =1&gt; void SubRS(xf::Mat&lt;SRC_T, ROWS, COLS, NPC&gt; &amp; _src1, unsigned char _scl[XF_CHANNELS(SRC_T,NPC)],xf::Mat&lt;SRC _T, ROWS, COLS, NPC&gt; &amp; _dst)</pre>



表 9: ビデオ処理関数 (続き)

関数	HLS ビデオ ライブラリ API	xfOpenCV ライブラリ API
Set	<pre>template&lt;int ROWS, int COLS, int SRC_T, typename T, int DST_T&gt; void Set(Mat&lt;ROWS, COLS, SRC_T&gt;&amp; src,         Scalar&lt;HLS_MAT_CN(SRC_T), T&gt;         scl,         Mat&lt;ROWS, COLS, DST_T&gt;&amp; dst)</pre>	<pre>template&lt; int SRC_T, int ROWS, int COLS, int NPC =1&gt; void set(xf::Mat&lt;SRC_T, ROWS, COLS, NPC&gt; &amp; _src1, unsigned char _src1[XF_CHANNELS(SRC_T,NPC)],xf::Mat&lt;SRC _T, ROWS, COLS, NPC&gt; &amp; _dst)</pre>
AbsDiff	<pre>template&lt;int ROWS, int COLS, int SRC1_T, int SRC2_T, int DST_T&gt; void AbsDiff(         Mat&lt;ROWS, COLS, SRC1_T&gt;&amp; src1,         Mat&lt;ROWS, COLS, SRC2_T&gt;&amp; src2,         Mat&lt;ROWS, COLS, DST_T&gt;&amp; dst)</pre>	<pre>template&lt;int SRC_T, int ROWS, int COLS, int NPC =1&gt; void absdiff(xf::Mat&lt;SRC_T, ROWS, COLS, NPC&gt; &amp; _src1,xf::Mat&lt;SRC_T, ROWS, COLS, NPC&gt; &amp; _src2,xf::Mat&lt;SRC_T, ROWS, COLS, NPC&gt; &amp; _dst)</pre>
And	<pre>template&lt;int ROWS, int COLS, int SRC1_T, int SRC2_T, int DST_T&gt; void And(         Mat&lt;ROWS, COLS, SRC1_T&gt;&amp; src1,         Mat&lt;ROWS, COLS, SRC2_T&gt;&amp; src2,         Mat&lt;ROWS, COLS, DST_T&gt;&amp; dst)</pre>	<pre>template&lt;int SRC_T, int ROWS, int COLS, int NPC = 1&gt; void bitwise_and(xf::Mat&lt;SRC_T, ROWS, COLS, NPC&gt; &amp; _src1, xf::Mat&lt;SRC_T, ROWS, COLS, NPC&gt; &amp; _src2, xf::Mat&lt;SRC_T, ROWS, COLS, NPC&gt; &amp; _dst)</pre>
Dilate	<pre>template&lt;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&gt; void Dilate(Mat&lt;IMG_HEIGHT, IMG_WIDTH, SRC_T&gt;&amp; src,Mat&lt;IMG_HEIGHT, IMG_WIDTH, DST_T&gt;&amp; dst,Window&lt;K_HEIGHT,K_WIDTH,KN_T&gt; &amp; _kernel)</pre>	<pre>template&lt;int BORDER_TYPE, int TYPE, int ROWS, int COLS,int K_SHAPE,int K_ROWS,int K_COLS, int ITERATIONS, int NPC=1&gt; void dilate (xf::Mat&lt;TYPE, ROWS, COLS, NPC&gt; &amp; _src, xf::Mat&lt;TYPE, ROWS, COLS, NPC&gt; &amp; _dst,unsigned char _kernel[K_ROWS*K_COLS])</pre>
Duplicate	<pre>template&lt;int ROWS, int COLS, int SRC_T, int DST_T&gt; void Duplicate(Mat&lt;ROWS, COLS, SRC_T&gt;&amp; src,Mat&lt;ROWS, COLS, DST_T&gt;&amp; dst1,Mat&lt;ROWS, COLS, DST_T&gt;&amp; dst2)</pre>	<pre>template&lt;int SRC_T, int ROWS, int COLS,int NPC&gt; void duplicateMat(xf::Mat&lt;SRC_T, ROWS, COLS, NPC&gt; &amp; _src, xf::Mat&lt;SRC_T, ROWS, COLS, NPC&gt; &amp; _dst1,xf::Mat&lt;SRC_T, ROWS, COLS, NPC&gt; &amp; _dst2)</pre>
EqualizeHist	<pre>template&lt;int SRC_T, int DST_T,int ROW, int COL&gt; void EqualizeHist(Mat&lt;ROW, COL, SRC_T&gt;&amp; _src,Mat&lt;ROW, COL, DST_T&gt;&amp; _dst)</pre>	<pre>template&lt;int SRC_T, int ROWS, int COLS, int NPC = 1&gt; void equalizeHist(xf::Mat&lt;SRC_T, ROWS, COLS, NPC&gt; &amp; _src,xf::Mat&lt;SRC_T, ROWS, COLS, NPC&gt; &amp; _src1,xf::Mat&lt;SRC_T, ROWS, COLS, NPC&gt; &amp; _dst)</pre>
erode	<pre>template&lt;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&gt; void Erode(Mat&lt;IMG_HEIGHT, IMG_WIDTH, SRC_T&gt;&amp; src,Mat&lt;IMG_HEIGHT, IMG_WIDTH, DST _T&gt;&amp; dst,Window&lt;K_HEIGHT,K_WIDTH,KN_T&gt;&amp; kernel)</pre>	<pre>template&lt;int BORDER_TYPE, int TYPE, int ROWS, int COLS,int K_SHAPE,int K_ROWS,int K_COLS, int ITERATIONS, int NPC=1&gt; void erode (xf::Mat&lt;TYPE, ROWS, COLS, NPC&gt; &amp; _src, xf::Mat&lt;TYPE, ROWS, COLS, NPC&gt; &amp; _dst,unsigned char _kernel[K_ROWS*K_COLS])</pre>

表 9: ビデオ処理関数 (続き)

関数	HLS ビデオ ライブラリ API	xfOpenCV ライブラリ API
FASTX	<pre>template&lt;int SRC_T,int ROWS,int COLS&gt; void FASTX(Mat&lt;ROWS,COLS,SRC_T&gt; &amp;_src, Mat&lt;ROWS,COLS,HLS_8UC1&gt;&amp;_mask,HLS_TNAME( SRC_T)_threshold,Bool _nmax_supression)</pre>	<pre>template&lt;int NMS,int SRC_T,int ROWS, int COLS,int NPC=1&gt; void fast(xf::Mat&lt;SRC_T, ROWS, COLS, NPC&gt; &amp;_src_mat,xf::Mat&lt;SRC_T, ROWS, COLS, NPC&gt; &amp;_dst_mat,unsigned char _threshold)</pre>
Filter2D	<pre>template&lt;int SRC_T, int DST_T, typename KN_T, typename POINT_T, int IMG_HEIGHT,int IMG_WIDTH,int K_HEIGHT,int K_WIDTH&gt; void Filter2D(Mat&lt;IMG_HEIGHT, IMG_WIDTH, SRC_T&gt; &amp;_src,Mat&lt;IMG_HEIGHT, IMG_WIDTH, DST_T&gt; &amp;_dst,Window&lt;K_HEIGHT,K_WIDTH,KN_T&gt;&amp;_ker nel,Point&lt;POINT_T&gt;_anchor)</pre>	<pre>template&lt;int BORDER_TYPE,int FILTER_WIDTH,int FILTER_HEIGHT, int SRC_T,int DST_T, int ROWS, int COLS,int NPC&gt; void filter2D(xf::Mat&lt;SRC_T, ROWS, COLS, NPC&gt; &amp;_src_mat,xf::Mat&lt;DST_T, ROWS, COLS, NPC&gt; &amp;_dst_mat,short int filter[FILTER_HEIGHT*FILTER_WIDTH],unsig ned char _shift)</pre>
GaussianBlur	<pre>template&lt;int KH,int KW,typename BORDERMODE,int SRC_T,int DST_T,int ROWS,int COLS&gt; void GaussianBlur(Mat&lt;ROWS, COLS, SRC_T&gt; &amp;_src, Mat&lt;ROWS, COLS, DST_T&gt; &amp;_dst,double sigmaX=0,double sigmaY=0)</pre>	<pre>template&lt;int FILTER_SIZE, int BORDER_TYPE, int SRC_T, int ROWS, int COLS,int NPC = 1&gt; void GaussianBlur(xf::Mat&lt;SRC_T, ROWS, COLS, NPC&gt; &amp;_src, xf::Mat&lt;SRC_T, ROWS, COLS, NPC&gt; &amp;_dst, float sigma)</pre>
Harris	<pre>template&lt;int blockSize,int Ksize,typename KT,int SRC_T,int DST_T,int ROWS,int COLS&gt; void Harris(Mat&lt;ROWS, COLS, SRC_T&gt; &amp;_src,Mat&lt;ROWS, COLS, DST_T&gt;&amp;_dst,KT k,int threshold)</pre>	<pre>template&lt;int FILTERSIZE,int BLOCKWIDTH, int NMSRADIUS,int SRC_T,int ROWS, int COLS,int NPC=1,bool USE_URAM=false&gt; void cornerHarris(xf::Mat&lt;SRC_T, ROWS, COLS, NPC&gt; &amp;_src,xf::Mat&lt;SRC_T, ROWS, COLS, NPC&gt; &amp;_dst,uint16_t threshold, uint16_t k)</pre>
CornerHarris	<pre>template&lt;int blockSize,int Ksize,typename KT,int SRC_T,int DST_T,int ROWS,int COLS&gt; void CornerHarris( Mat&lt;ROWS, COLS, SRC_T&gt;&amp;_src,Mat&lt;ROWS, COLS, DST_T&gt;&amp;_dst,KT k)</pre>	<pre>template&lt;int FILTERSIZE,int BLOCKWIDTH, int NMSRADIUS,int SRC_T,int ROWS, int COLS,int NPC=1,bool USE_URAM=false&gt; void cornerHarris(xf::Mat&lt;SRC_T, ROWS, COLS, NPC&gt; &amp;_src,xf::Mat&lt;SRC_T, ROWS, COLS, NPC&gt; &amp;_dst,uint16_t threshold, uint16_t k)</pre>
HoughLines2	<pre>template&lt;unsigned int theta,unsigned int rho,typename AT,typename RT,int SRC_T,int ROW,int COL,unsigned int linesMax&gt; void HoughLines2(Mat&lt;ROW,COL,SRC_T&gt; &amp;_src, Polar&lt;AT,RT&gt; (&amp;_lines) [linesMax],unsigned int threshold)</pre>	<pre>template&lt;unsigned int RHO,unsigned int THETA,int MAXLINES,int DIAG,int MINTHETA,int MAXTHETA,int SRC_T, int ROWS, int COLS,int NPC&gt; void HoughLines(xf::Mat&lt;SRC_T, ROWS, COLS, NPC&gt; &amp;_src_mat,float outputrho[MAXLINES],float outputtheta[MAXLINES],short threshold,short linesmax)</pre>
Integral	<pre>template&lt;int SRC_T, int DST_T, int ROWS,int COLS&gt; void Integral(Mat&lt;ROWS, COLS, SRC_T&gt;&amp;_src, Mat&lt;ROWS+1, COLS+1, DST_T&gt;&amp;_sum )</pre>	<pre>template&lt;int SRC_TYPE,int DST_TYPE, int ROWS, int COLS, int NPC&gt; void integral(xf::Mat&lt;SRC_TYPE, ROWS, COLS, NPC&gt; &amp;_src_mat, xf::Mat&lt;DST_TYPE, ROWS, COLS, NPC&gt; &amp; _dst_mat)</pre>

表 9: ビデオ処理関数 (続き)

関数	HLS ビデオ ライブラリ API	xfOpenCV ライブラリ API
Merge	<pre>template&lt;int ROWS, int COLS, int SRC_T, int DST_T&gt; void Merge(     Mat&lt;ROWS, COLS, SRC_T&gt;&amp; src0,     Mat&lt;ROWS, COLS, SRC_T&gt;&amp; src1,     Mat&lt;ROWS, COLS, SRC_T&gt;&amp; src2,     Mat&lt;ROWS, COLS, SRC_T&gt;&amp; src3,     Mat&lt;ROWS, COLS, DST_T&gt;&amp; dst)</pre>	<pre>template&lt;int SRC_T, int DST_T, int ROWS, int COLS, int NPC=1&gt; void merge(xf::Mat&lt;SRC_T, ROWS, COLS, NPC&gt; &amp;_src1, xf::Mat&lt;SRC_T, ROWS, COLS, NPC&gt; &amp;_src2, xf::Mat&lt;SRC_T, ROWS, COLS, NPC&gt; &amp;_src3, xf::Mat&lt;SRC_T, ROWS, COLS, NPC&gt; &amp;_src4, xf::Mat&lt;DST_T, ROWS, COLS, NPC&gt; &amp;_dst)</pre>
MinMaxLoc	<pre>template&lt;int ROWS, int COLS, int SRC_T, typename P_T&gt; void MinMaxLoc(Mat&lt;ROWS, COLS, SRC_T&gt;&amp; src, P_T* min_val, P_T* max_val, Point&amp; min_loc, Point&amp; max_loc)</pre>	<pre>template&lt;int SRC_T, int ROWS, int COLS, int NPC=0&gt; void minMaxLoc(xf::Mat&lt;SRC_T, ROWS, COLS, NPC&gt; &amp;_src, int32_t *min_value, int32_t *max_value, uint16_t *minlocx, uint16_t *minlocy, uint16_t *maxlocx, uint16_t *maxlocy)</pre>
Mul	<pre>template&lt;int ROWS, int COLS, int SRC1_T, int SRC2_T, int DST_T&gt; void Mul(Mat&lt;ROWS, COLS, SRC1_T&gt;&amp; src1, Mat&lt;ROWS, COLS, SRC2_T&gt;&amp; src2, Mat&lt;ROWS, COLS, DST_T&gt;&amp; dst)</pre>	<pre>template&lt;int POLICY_TYPE, int SRC_T, int ROWS, int COLS, int NPC = 1&gt; void multiply(xf::Mat&lt;SRC_T, ROWS, COLS, NPC&gt; &amp; src1, xf::Mat&lt;SRC_T, ROWS, COLS, NPC&gt; &amp; src2, xf::Mat&lt;SRC_T, ROWS, COLS, NPC&gt; &amp; dst, float scale)</pre>
Not	<pre>template&lt;int ROWS, int COLS, int SRC_T, int DST_T&gt; void Not(Mat&lt;ROWS, COLS, SRC_T&gt;&amp; src, Mat&lt;ROWS, COLS, DST_T&gt;&amp; dst)</pre>	<pre>template&lt;int SRC_T, int ROWS, int COLS, int NPC = 1&gt; void bitwise_not(xf::Mat&lt;SRC_T, ROWS, COLS, NPC&gt; &amp; src, xf::Mat&lt;SRC_T, ROWS, COLS, NPC&gt; &amp; dst)</pre>
Range	<pre>template&lt;int ROWS, int COLS, int SRC_T, int DST_T, typename P_T&gt; void Range(Mat&lt;ROWS, COLS, SRC_T&gt;&amp; src, Mat&lt;ROWS, COLS, DST_T&gt;&amp; dst, P_T start, P_T end)</pre>	<pre>template&lt;int SRC_T, int ROWS, int COLS, int NPC=1&gt; void inRange(xf::Mat&lt;SRC_T, ROWS, COLS, NPC&gt; &amp; src, unsigned char lower_thresh, unsigned char upper_thresh, xf::Mat&lt;SRC_T, ROWS, COLS, NPC&gt; &amp; dst)</pre>
Resize	<pre>template&lt;int SRC_T, int ROWS, int COLS, int DROWS, int DCOLS&gt; void Resize(     Mat&lt;ROWS, COLS, SRC_T&gt; &amp; src,     Mat&lt;DROWS, DCOLS, SRC_T&gt; &amp; dst,     int interpolation=HLS_INTER_LINEAR)</pre>	<pre>template&lt;int INTERPOLATION_TYPE, int TYPE, int SRC_ROWS, int SRC_COLS, int DST_ROWS, int DST_COLS, int NPC, int MAX_DOWN_SCALE&gt; void resize(xf::Mat&lt;TYPE, SRC_ROWS, SRC_COLS, NPC&gt; &amp; src, xf::Mat&lt;TYPE, DST_ROWS, DST_COLS, NPC&gt; &amp; dst)</pre>
sobel	<pre>template&lt;int XORDER, int YORDER, int SIZE, int SRC_T, int DST_T, int ROWS, int COLS, int DROWS, int DCOLS&gt; void Sobel(Mat&lt;ROWS, COLS, SRC_T&gt; &amp;_src, Mat&lt;DROWS, DCOLS, DST_T&gt; &amp;_dst)</pre>	<pre>template&lt;int BORDER_TYPE, int FILTER_TYPE, int SRC_T, int DST_T, int ROWS, int COLS, int NPC=1, bool USE_URAM = false&gt; void Sobel(xf::Mat&lt;SRC_T, ROWS, COLS, NPC&gt; &amp;_src_mat, xf::Mat&lt;DST_T, ROWS, COLS, NPC&gt; &amp;_dst_mat, xf::Mat&lt;DST_T, ROWS, COLS, NPC&gt; &amp;_dst_maty)</pre>

表 9: ビデオ処理関数 (続き)

関数	HLS ビデオ ライブラリ API	xfOpenCV ライブラリ API
split	<pre>template&lt;int ROWS, int COLS, int SRC_T, int DST_T&gt; void Split(     Mat&lt;ROWS, COLS, SRC_T&gt;&amp; src,     Mat&lt;ROWS, COLS, DST_T&gt;&amp; dst0,     Mat&lt;ROWS, COLS, DST_T&gt;&amp; dst1,     Mat&lt;ROWS, COLS, DST_T&gt;&amp; dst2,     Mat&lt;ROWS, COLS, DST_T&gt;&amp; dst3)</pre>	<pre>template&lt;int SRC_T, int DST_T, int ROWS, int COLS, int NPC=1&gt; void extractChannel(xf::Mat&lt;SRC_T, ROWS, COLS, NPC&gt; &amp; _src_mat, xf::Mat&lt;DST_T, ROWS, COLS, NPC&gt; &amp; _dst_mat, uint16_t _channel)</pre>
Threshold	<pre>template&lt;int ROWS, int COLS, int SRC_T, int DST_T&gt; void Threshold(     Mat&lt;ROWS, COLS, SRC_T&gt;&amp; src,     Mat&lt;ROWS, COLS, DST_T&gt;&amp; dst,     HLS_TNAME(SRC_T) thresh,     HLS_TNAME(DST_T) maxval,     int thresh_type)</pre>	<pre>template&lt;int THRESHOLD_TYPE, int SRC_T, int ROWS, int COLS, int NPC=1&gt; void Threshold(xf::Mat&lt;SRC_T, ROWS, COLS, NPC&gt; &amp; _src_mat, xf::Mat&lt;SRC_T, ROWS, COLS, NPC&gt; &amp; _dst_mat, short int thresh, short int maxval)</pre>
Scale	<pre>template&lt;int ROWS, int COLS, int SRC_T, int DST_T, typename P_T&gt; void Scale(Mat&lt;ROWS, COLS, SRC_T&gt;&amp; src, Mat&lt;ROWS, COLS, DST_T&gt;&amp; dst, P_T scale=1.0, P_T shift=0.0)</pre>	<pre>template&lt; int SRC_T, int DST_T, int ROWS, int COLS, int NPC = 1&gt; void scale(xf::Mat&lt;SRC_T, ROWS, COLS, NPC&gt; &amp; src1, xf::Mat&lt;DST_T, ROWS, COLS, NPC&gt; &amp; dst, float scale, float shift)</pre>
InitUndistortRectifyMapInverse	<pre>template&lt;typename CMT, typename DT, typename ICMT, int ROWS, int COLS, int MAP1_T, int MAP2_T, int N&gt; void InitUndistortRectifyMapInverse (     Window&lt;3,3, CMT&gt; cameraMatrix, DT(&amp;distCoeffs) [N], Window&lt;3,3, ICMT&gt; ir, Mat&lt;ROWS, COLS, MAP1_T&gt; &amp;map1, Mat&lt;ROWS, COLS, MAP2_T&gt; &amp;map2, int noRotation=false)</pre>	<pre>template&lt; int CM_SIZE, int DC_SIZE, int MAP_T, int ROWS, int COLS, int NPC &gt; void InitUndistortRectifyMapInverse (     ap_fixed&lt;32,12&gt; *cameraMatrix,     ap_fixed&lt;32,12&gt; *distCoeffs,     ap_fixed&lt;32,12&gt; *ir,     xf::Mat&lt;MAP_T, ROWS, COLS, NPC&gt; &amp; mapx_mat, xf::Mat&lt;MAP_T, ROWS, COLS, NPC&gt; &amp; mapy_mat, int _cm_size, int _dc_size)</pre>
Avg、mean、AvgStddev	<pre>template&lt;typename DST_T, int ROWS, int COLS, int SRC_T&gt; DST_T Mean(Mat&lt;ROWS, COLS, SRC_T&gt;&amp; src)</pre>	<pre>template&lt;int SRC_T, int ROWS, int COLS, int NPC=1&gt; void meanStdDev(xf::Mat&lt;SRC_T, ROWS, COLS, NPC&gt; &amp; _src, unsigned short* _mean, unsigned short* _stddev)</pre>
CvtColor	<pre>template&lt;typename CONVERSION, int SRC_T, int DST_T, int ROWS, int COLS&gt; void CvtColor(Mat&lt;ROWS, COLS, SRC_T&gt; &amp; _src,     Mat&lt;ROWS, COLS, DST_T&gt; &amp; _dst)</pre>	色変換

注記: Reduce 以外のすべての関数では、1 クロックごとに N ピクセル (N は 2 のべき乗) を処理できます。

## xfOpenCV ライブラリの使用

このセクションでは、SDx 開発環境での xfOpenCV ライブラリの使用方法を説明します。

**注記:** このセクションの手順は、必要なパッケージをすべてダウンロードおよびインストールしていることを前提としています。詳細は、[使用要件](#)を参照してください。

include フォルダーには、コンピューター ビジョンまたは画像処理パイプラインを構築するのに必要なすべてのコンポーネントが含まれます。common および core フォルダーには、ライブラリ関数で基本的な関数、Mat クラス、およびマクロに必要なインフラストラクチャが含まれます。ライブラリ関数は、実行する操作に応じて features、video、および imgproc の3つのフォルダーに分類されています。フォルダー名はわかりやすいものになっています。

ライブラリ関数を使用するには、SDx プロジェクトに include フォルダーへのパスを含める必要があります。include フォルダーのパスをコンパイラに読み込むと、使用するライブラリ関数に関連するヘッダー ファイルを含めることができますようになります。たとえば Harris コーナー検出器およびバイラテラル フィルターを使用するには、ホスト コードに次の行を含める必要があります。

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

ヘッダーを含めたら、[第5章: xfOpenCV ライブラリ API リファレンス](#)に説明されているように、ライブラリを使用できます。examples フォルダーの例を参考にしてください。

次の表に、ヘッダー ファイルの名前と、ライブラリ関数を含むフォルダー名を示します。

表 10: xfOpenCV ライブラリの内容

関数名	include 内のファイルパス
xf::accumulate	imgproc/xf_accumulate_image.hpp
xf::accumulateSquare	imgproc/xf_accumulate_squared.hpp
xf::accumulateWeighted	imgproc/xf_accumulate_weighted.hpp
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

表 10: xfOpenCV ライブラリの内容 (続き)

関数名	include 内のファイルパス
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::uyvy2iyuv、xf::uyvy2nv12、xf::uyvy2rgba、xf::rgb2iyuv、xf::rgb2nv12、xf::rgb2nv21、xf::rgb2yuv4、xf::rgb2uyvy、xf::rgb2uyyv、xf::rgb2bgr、xf::bgr2uyvy、xf::bgr2uyyv、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
xf::integrallImage	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	imgproc/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

表 10: xfOpenCV ライブラリの内容 (続き)

関数名	include 内のファイルパス
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

xfOpenCV ライブラリの例を使用するには、次の方法があります。

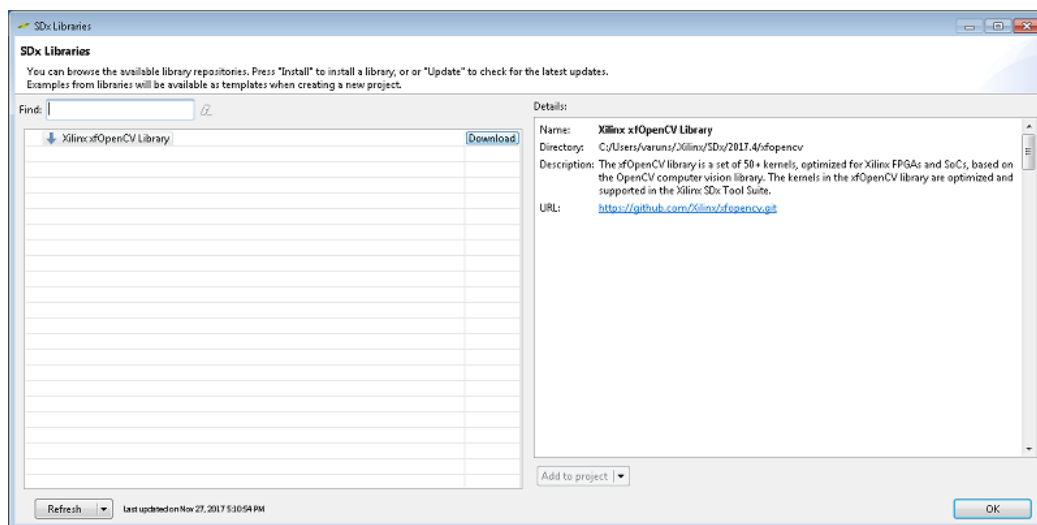
- SDx GUI から xfOpenCV ライブラリをダウンロードして使用方法
- サンプル makefile を使用した Linux でのプロジェクトの構築
- reVISION プラットフォームでの reVISION サンプルの使用
- reVISION 以外のプラットフォームでの xfOpenCV ライブラリの使用

## SDx GUI から xfOpenCV ライブラリをダウンロードして使用方法

xfOpenCV は SDx GUI から直接ダウンロードできます。Linux プラットフォームのサンプル makefile を使用してプロジェクトをビルドするには、次の手順に従います。

1. SDx IDE で [Xilinx] → [SDx Libraries] をクリックします。
2. [Xilinx xfOpenCV Library] の横の [Download] ボタンをクリックします。

図 2: SDx ライブラリ



ライブラリは <home directory>/Xilinx/SDx/2019.1/xfopencv にダウンロードされます。ライブラリがダウンロードされると、新しいプロジェクトを作成するときにテンプレートのリストにライブラリに含まれるすべての例が表示されます。

**注記:** ライブラリは、IDE メニュー オプションからどのプロジェクトにも追加できます。

- プロジェクトにライブラリを追加するには、SDx IDE で [Xilinx] → [SDx Libraries] をクリックします。
- [Xilinx xfOpenCV Library] を選択し、[Add to project] をクリックします。ドロップダウンリストに、ライブラリを追加する必要のあるプロジェクトのリストが表示されます。

xfOpenCV ライブラリの include/ フォルダに含まれるすべてのヘッダーが、ローカル プロジェクト ディレクトリの <project\_dir>/libs/xfopencv/include にコピーされます。この操作を実行すると、実行するライブラリに必要なすべての設定も指定されます。

## サンプル makefile を使用した Linux でのプロジェクトの構築

Linux プラットフォームでサンプル makefile を使用してプロジェクトを構築するには、次の手順に従います。

- ターミナルを開きます。
- reVISION プラットフォームをビルドする場合は、環境変数 SYSROOT を <the path to platform folder>/sw/a53\_linux/a53\_linux/sysroot/aarch64-xilinx-linux に設定します。
- プラットフォーム変数を makefile のダウンロード済みプラットフォーム フォルダをポイントするように変更します。ダウンロード済みプラットフォームのフォルダ名が変更されていないことを確認してください。
- reVISION プラットフォームをビルドする場合は、makefile の IDIRS および LDIRS 変数を次のように変更します。

```
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
```

- 例をビルドするディレクトリに移動します。

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

- reVISION プラットフォームをビルドする場合は、xf\_headers.h ファイルの if および else 部分の両方に #include "opencv2/imgcodecs/imgcodecs.hpp" を追加します。
- SDx 開発環境を実行する環境変数を設定します。

- C シェル:

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

- bash シェル:

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

- ターミナルに make コマンドを入力します。sd\_card フォルダが <path to example> に作成されます。

**注記:** reVISION 以外のプラットフォームをビルドする際は、手順 2、4、および 6 は無視します。

## reVISION プラットフォームでの reVISION サンプルの使用

zcu104\_rv\_ss でバイラテラル フィルターのユニットテストを実行するには、次の手順に従います。

- デスクトップアイコンをダブルクリックするか[スタート]メニューを使用して、SDx 開発環境を起動します。



[Workspace Launcher] ダイアログ ボックスが表示されます。

2. [Browse] をクリックしてプロジェクトを保存するワークスペース フォルダを選択し、[OK] をクリックします。

**注記:** Linux で SDx IDE を起動する前に、\$SYSROOT 環境変数を設定したのと同じシェルを使用していることを確認します。これは通常、Linux ルート ファイル システムへのファイルパスです。

SDx 開発環境のメインウィンドウが表示されます。新しいワークスペースを作成した場合は、[Welcome] タブが表示されます。[Welcome] タブは、[X] をクリックして閉じるか、[Minimize] アイコンをクリックして最小化できます。

3. SDx 開発環境のメニュー バーから [File] → [New] → [Xilinx SDx Project] をクリックします。

[New Project] ダイアログ ボックスが開きます。

4. プロジェクト名を指定します (「Bilateral」 など)。

5. [Next] をクリックします。

[Choose Hardware Platform] ページが開きます。

6. [Choose Hardware Platform] ページで [Add Custom Platform] をクリックします。

7. reVISION プラットフォーム ファイルを解凍したディレクトリに移動します。zcu104\_rv\_ss フォルダを選択します。

8. [Choose Hardware Platform] ページで [zcu104\_rv\_ss (custom)] を選択します。

9. [Next] をクリックします。

選択したプラットフォーム用のソース コード例をリストする [Templates] ページが表示されます。

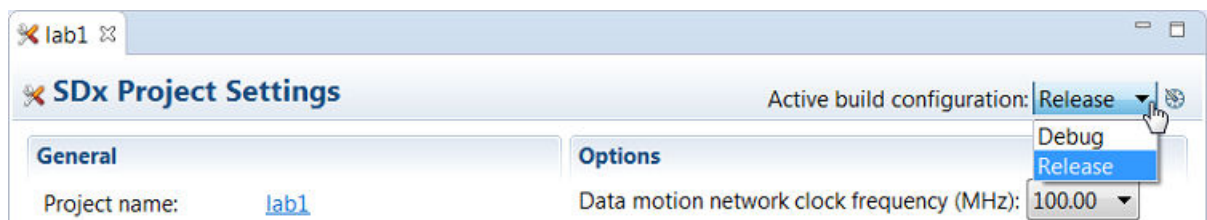
10. [Available Templates] から [bilateral - File I/O] を選択し、[Finish] をクリックします。

11. project/src/examples/bilateral/ にある xf\_headers.h ファイルの if および else 部分の両方に #include "opencv2/imgcodecs/imgcodecs.hpp" を追加します。

12. [SDx Project Settings] で [Active build configurations] ドロップダウン リストをクリックしてアクティブ コンフィギュレーションを選択するか、ビルド コンフィギュレーションを作成します。

標準ビルド コンフィギュレーションは [Debug] および [Release] です。最高のランタイム パフォーマンスを得るには、[Release] ビルド コンフィギュレーションを選択します。このビルド コンフィギュレーションでは、Debug ビルド コンフィギュレーションよりも高いコンパイラ最適化設定が使用されます。

図 3: [SDx Project Settings] - [Active build configuration] ドロップダウン リスト



13. [SDx Project Settings] で [Data motion network clock frequency (MHz)] を必要な周波数に設定します。

14. [Project Explorer] ビューでプロジェクトを右クリックして [Build Project] をクリックするか、Ctrl + B キーを押してプロジェクトをビルドします。

15. 作成された sd\_card フォルダの内容を SD カードにコピーします。sd\_card フォルダには、ZCU104 ボードでデザインを実行するのに必要なファイルがすべて含まれます。

16. ZCU104 ボードのカード スロットに SD カードを挿入し、スイッチをオンにします。

**注記:** ユーザー コマンドをボードに渡すには、シリアル ポート エミュレーター (Teraterm/Minicom) が必要です。

17. 正しくブートしたら、Teraterm ターミナル(シリアル ポート エミュレーター)で次のコマンドを実行します。

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

18. 対応する関数の .elf ファイルを実行します。

詳細は、[xfOpenCV ライブラリ関数のハードウェアでの使用](#)を参照してください。

## reVISION 以外のプラットフォームでの xfOpenCV ライブラリの使用

このセクションでは、SDx™ 開発環境で reVISION 以外のプラットフォームに xfOpenCV ライブラリを使用する方法を説明します。xfOpenCV の例を正しくコンパイルするには、OpenCV ライブラリが必要です。reVISION 以外のプラットフォームでは OpenCV ライブラリを含めることは前提条件ではないので、互換性のある libjpeg.so を使用して OpenCV ライブラリをインストールおよびコンパイルする必要があります。

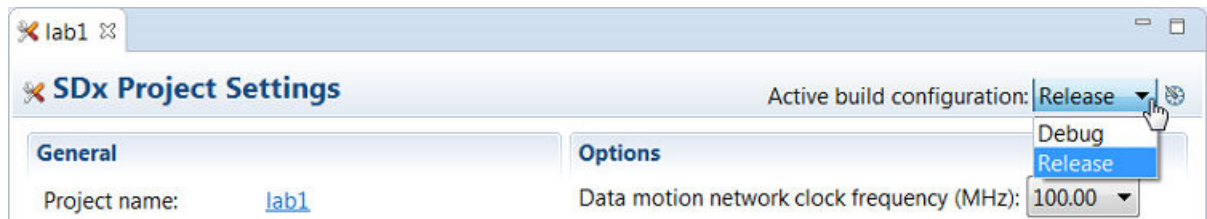
**注記:** このセクションの手順は、必要なパッケージをすべてダウンロードおよびインストールしていることを前提としています。詳細は、[使用要件](#)を参照してください。

xfOpenCV ライブラリを SDx プロジェクトにインポートしてカスタム プラットフォームで実行するには、次の手順に従います。

1. デスクトップアイコンをダブルクリックするか[スタート]メニューを使用して、SDx 開発環境を起動します。  
[Workspace Launcher] ダイアログ ボックスが表示されます。
2. [Browse] をクリックしてプロジェクトを保存するワークスペース フォルダーを選択し、[OK] をクリックします。  
SDx 開発環境のメイン ウィンドウが表示されます。新しいワークスペースを作成した場合は、[Welcome] タブが表示されます。[Welcome] タブは、[X] をクリックして閉じるか、[Minimize] アイコンをクリックして最小化できます。
3. SDx 開発環境のメニュー バーから [File] → [New] → [Xilinx SDx Project] をクリックします。  
[New Project] ダイアログ ボックスが開きます。
4. プロジェクト名を指定します(「Test」など)。
5. [Next] をクリックします。  
[Choose Hardware Platform] ページが開きます。
6. [Choose Hardware Platform] ページでプラットフォーム (zcu102 など) を選択します。
7. [Next] をクリックします。  
[Choose Software Platform and Target CPU] ページが表示されます。
8. [Choose Software Platform and Target CPU] ページで、ソフトウェア プラットフォームおよびターゲット CPU を選択します。たとえば、[CPU] ドロップダウン リストから ZC702 および ZC706 reVISION プラットフォーム用に [A9] を選択します。
9. [Next] をクリックします。選択したプラットフォーム用のソース コード例をリストする [Templates] ページが表示されます。
10. [Available Templates] から [Empty Application] を選択し、[Finish] をクリックします。  
[New Project] ダイアログ ボックスが閉じます。新しいプロジェクトが指定した設定で作成されます。[SDx Project Settings] が表示されます。このパネルの右下に進捗状況バーがあります。C/C++ Indexer が終了するのを待ちます。

11. 標準ビルド コンフィギュレーションは [Debug] および [Release] です。最高のランタイム時にタイム パフォーマンスを得るには、[Release] ビルド コンフィギュレーションを選択します。このビルド コンフィギュレーションでは、Debug ビルド コンフィギュレーションよりも高いコンパイラ最適化設定が使用されます。


図 4: [SDx Project Settings] - [Active build configuration] ドロップダウン リスト



12. [SDx Project Settings] で [Data motion network clock frequency (MHz)] を必要な周波数に設定します。
13. [Generate bitstream] および [Generate SD card Image] チェック ボックスをオンにします。
14. [Project Explorer] ビューで新しく作成したプロジェクトを右クリックします。
15. [C/C++ Build Settings] をクリックします。

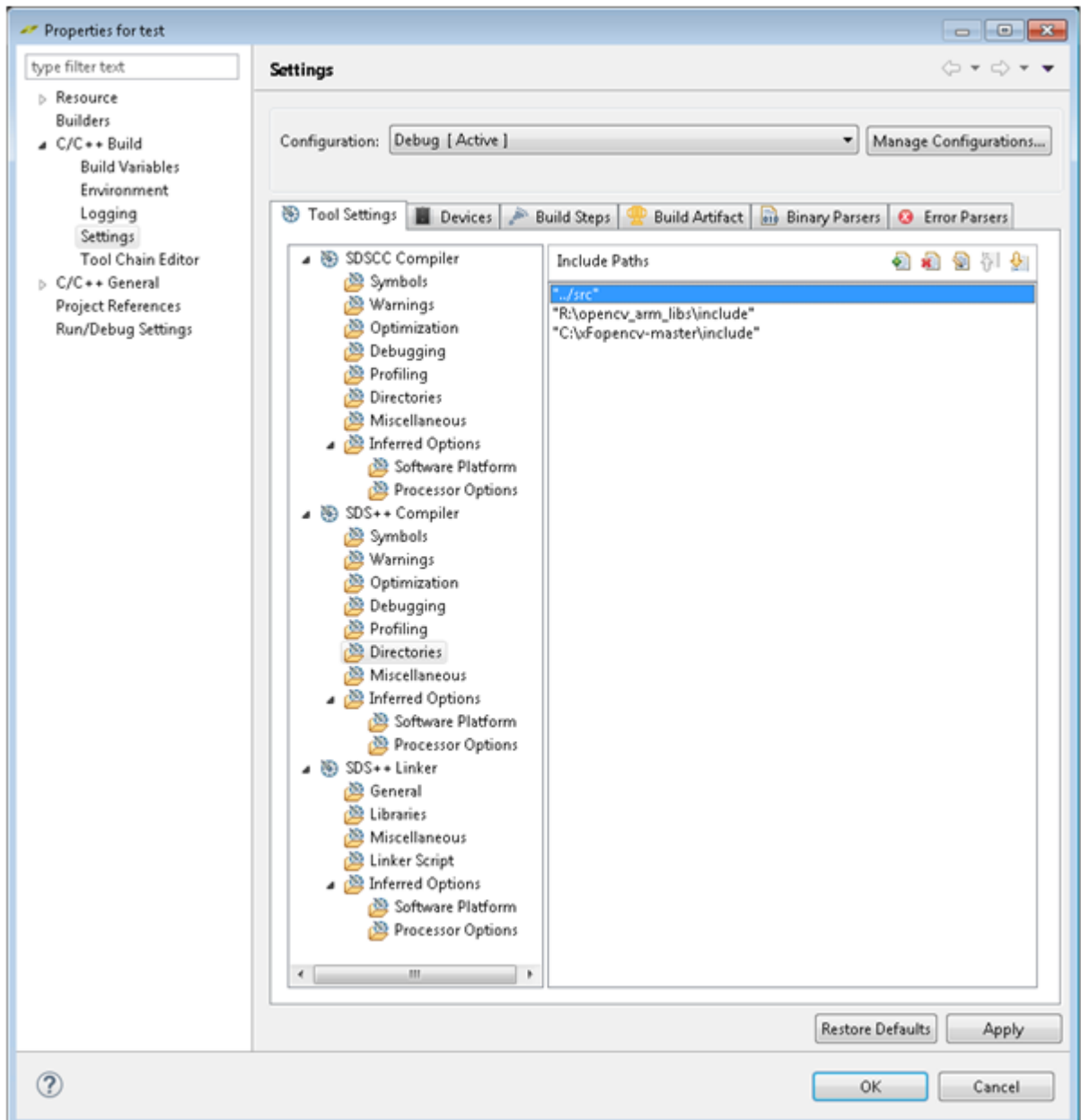
[Properties for <project>] ダイアログ ボックスが表示されます。


16. [Tool Settings] タブをクリックします。
17. [SDS++ Compiler] → [Directories] を展開します。

18.  アイコンをクリックして [Include Paths] リストに "<xfopencv\_location>\include" および "<opencv\_location>\include" フォルダを追加します。


**注記:** カスタム プラットフォームに対しては、サイリンクスにより OpenCV ザライブラリは提供されません。ユーザーがライブラリを提供する必要があります。サイリンクスが提供する OpenCV ライブラリを使用するには、reVISION プラットフォームを使用してください。

図 5: SDS++ コンパイラ設定



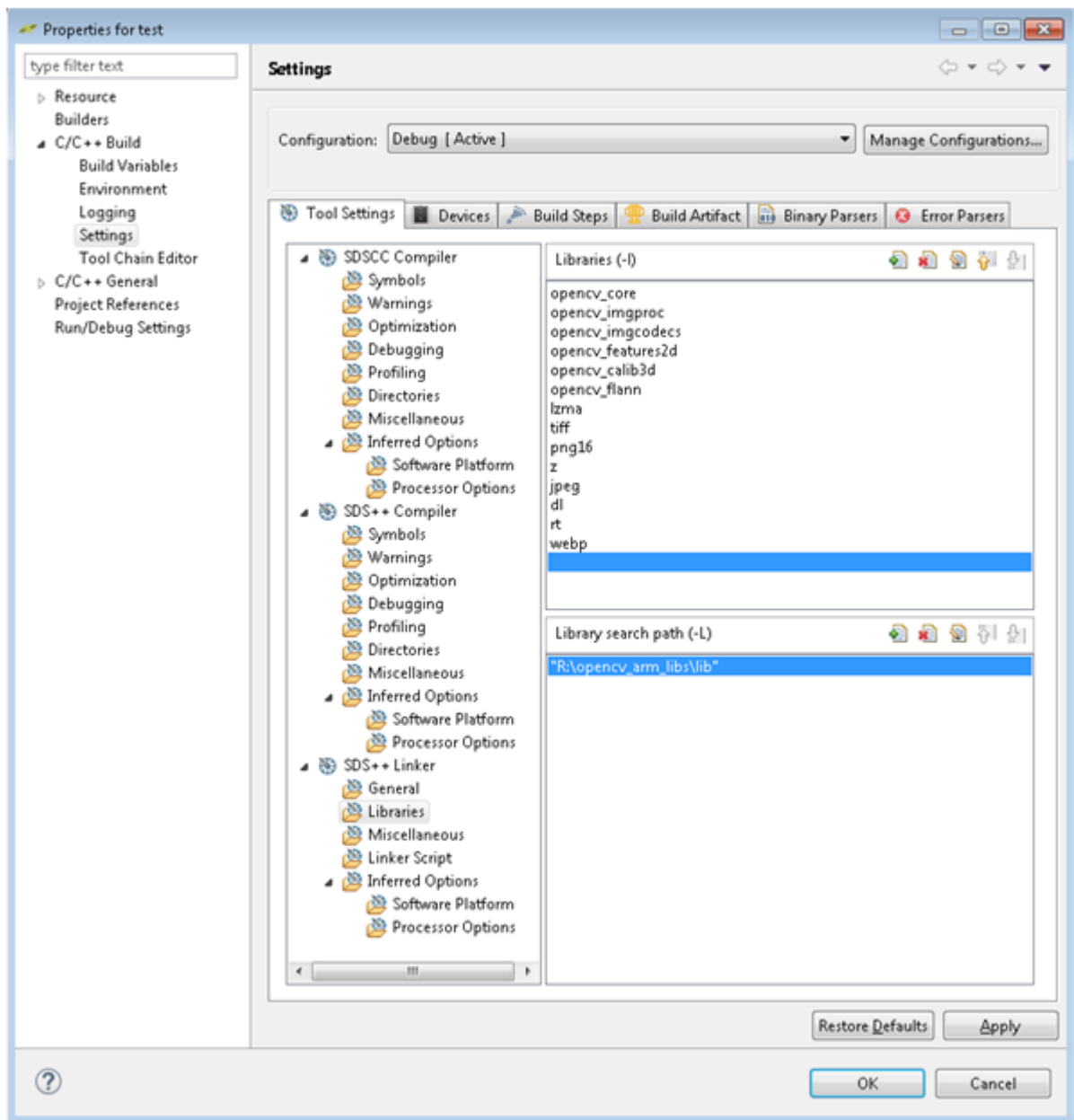
19. 同じページで [SDS++ Compiler] → [Inferred Options] → [Software Platform] を選択し、[Software Platform Inferred Flags] フィールドで「-hls-target 1」を指定します。
20. [Apply] をクリックします。
21. [SDS++ Linker] → [Libraries] を展開します。
22.  アイコンをクリックして [Libraries(-l)] リストに次のライブラリを追加します。これらのライブラリは OpenCV が必要です。
  - opencv\_core
  - opencv\_imgproc

- opencv\_imgcodecs
- opencv\_features2d
- opencv\_calib3d
- opencv\_flann
- opencv\_video
- opencv\_videoio

23.  アイコンをクリックして [Libraries search path (-L)] リストに <opencv\_Location>/lib フォルダーを追加します。

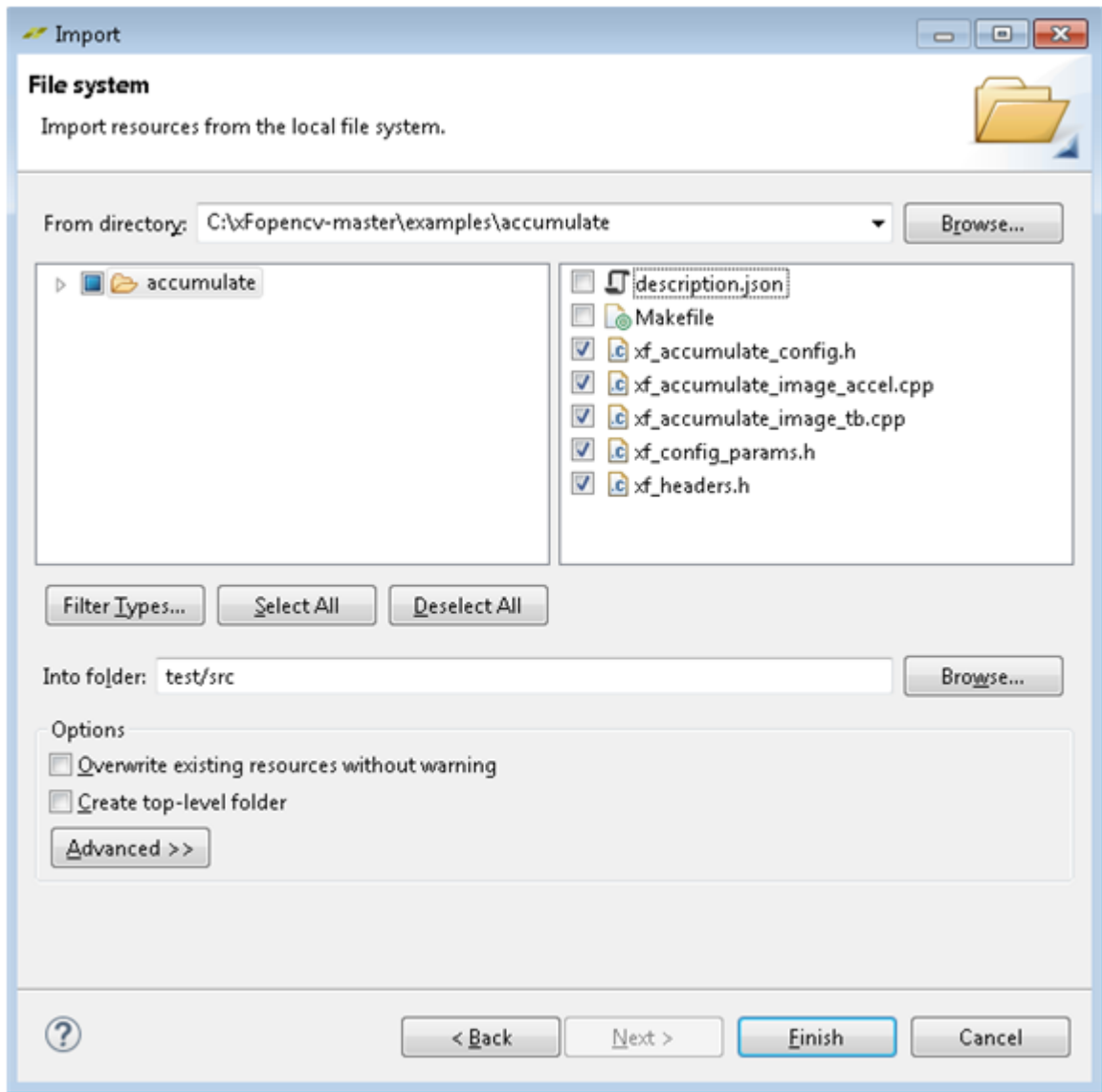
**注記:** カスタム プラットフォームに対しては、ザイリンクスにより OpenCV サライブラリは提供されません。ユーザーがライブラリを提供する必要があります。ザイリンクスが提供する OpenCV ライブラリを使用するには、reVISION プラットフォームを使用してください。

図 6: SDS++ リンカー設定



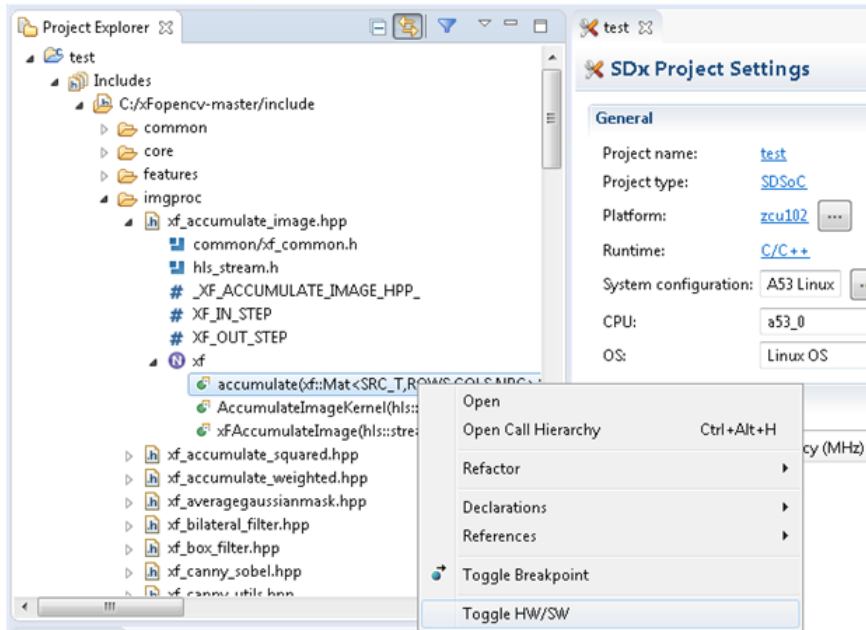
24. [Apply] をクリックして設定を保存します。
25. [OK] をクリックして [Properties for <project>] ダイアログ ボックスを閉じます。
26. [Project Explorer] ビューで新しく作成したプロジェクトを展開します。
27. src フォルダを右クリックし、[Import] をクリックします。[Import] ダイアログ ボックスが開きます。
28. [File System] を選択して [Next] をクリックします。
29. [Browse] をクリックして <xfoencv\_Location>/examples フォルダに移動します。
30. インポートするライブラリに対応するフォルダ (accumulate など) を選択します。

図 7: ライブラリのサンプルソースファイルのインポート



31. [Project Explorer] ビューでライブラリ関数を右クリックし、[Toggle HW/SW] をクリックして関数をハードウェアに移動します。

図 8: ライブラリ関数のハードウェアへの移動



32. [Project Explorer] ビューでプロジェクトを右クリックして [Build Project] をクリックするか、Ctrl + B キーを押してプロジェクトをビルドします。

ビルド プロセスは、ホスト マシンの処理能力およびデザインの複雑性によって数分から数時間かかります。ほとんどの時間は、ハードウェアに配置するよう指定されたルーチンの処理に費やされます。

33. 作成された `.\<workspace>\<function>\Release\sd_card` フォルダの内容を SD カードにコピーします。sd\_card フォルダには、ボードでデザインを実行するのに必要なファイルがすべて含まれます。
34. ボードのカード スロットに SD カードを挿入し、スイッチをオンにします。

**注記:** ユーザー コマンドをボードに渡すには、シリアルポート エミュレーター (Teraterm/Minicom) が必要です。

35. 正しくブートしたら、`./mnt` フォルダに移動してプロンプトで次のコマンドを実行します。

```
#cd /mnt
```

**注記:** OpenCV ライブラリはルート ファイルシステムのポートであると想定されます。そうでない場合は、`$ export LD_LIBRARY_PATH=<location of OpenCV libraries>/lib` コマンドを使用して OpenCV ライブラリの場所を LD\_LIBRARY\_PATH に追加します。

36. `.elf` 実行ファイルを実行します。詳細は、[xfOpenCV ライブラリ関数のハードウェアでの使用](#) を参照してください。

## ハードウェア カーネル コンフィギュレーションの変更

ハードウェア カーネル コンフィギュレーションを変更するには、次の手順に従います。

1. `<path to xfOpenCV git folder>/xfOpenCV/examples/<function>/xf_config_params.h` ファイルをアップデートします。
2. `makefile` と `xf_config_params.h` ファイルをアップデートします。



- a. makefile で関数名を含む行を見つけます。バイラテラルフィルタの場合、makefile 内の行は `xf::BilateralFilter<3,1,0,1080,1920,1>` です。
- b. `xf_config_params.h` ファイルに加えた変更を反映するために makefile のテンプレートパラメーターをアップデートします。詳細は、[第 5 章: xfOpenCV ライブラリ API リファレンス](#) を参照してください。

## xfOpenCV ライブラリ関数のハードウェアでの使用

次の表に、xfOpenCV ライブラリ関数と、それらに対応する例をハードウェアで実行するコマンドを示します。デザインが完全にビルドされており、ボードが正しく起動されている必要があります。

表 11: xfOpenCV ライブラリ関数のハードウェアでの使用

例	関数名	ハードウェアでの使用方法
accumulate	xf::accumulate	./<実行ファイル名>.elf <入力画像 1 へのパス> <入力画像 2 へのパス>
accumulatesquared	xf::accumulateSquare	./<実行ファイル名>.elf <入力画像 1 へのパス> <入力画像 2 へのパス>
accumulateweighted	xf::accumulateWeighted	./<実行ファイル名>.elf <入力画像 1 へのパス> <入力画像 2 へのパス>
addS	xf::addS	./<実行ファイル名>.elf <入力画像へのパス>
arithm	xf::absdiff、xf::add、xf::subtract、 xf::bitwise_and、xf::bitwise_or、xf::bitwise_not、 xf::bitwise_xor	./<実行ファイル名>.elf <入力画像 1 へのパス> <入力画像 2 へのパス>
addweighted	xf::addWeighted	./<実行ファイル名>.elf <入力画像 1 へのパス> <入力画像 2 へのパス>
Bilateralfilter	xf::bilateralFilter	./<実行ファイル名>.elf <入力画像へのパス>
Boxfilter	xf::boxFilter	./<実行ファイル名>.elf <入力画像へのパス>
Boundingbox	xf::boundingbox	./<実行ファイル名>.elf <入力画像へのパス> <ROI の数>
Canny	xf::Canny	./<実行ファイル名>.elf <入力画像へのパス>
channelcombine	xf::merge	./<実行ファイル名>.elf <入力画像 1 へのパス> <入力画像 2 へのパス> <入力画像 3 へのパス> <入力画像 4 へのパス>
Channelextract	xf::extractChannel	./<実行ファイル名>.elf <入力画像へのパス>
Colordetect	xf::bgr2hsv、xf::colorthresholding、xf::erode、 および xf::dilate	./<実行ファイル名>.elf <入力画像へのパス>
compare	xf::compare	./<実行ファイル名>.elf <入力画像 1 へのパス> <入力画像 2 へのパス>
compareS	xf::compareS	./<実行ファイル名>.elf <入力画像へのパス>
Convertbitdepth	xf::convertTo	./<実行ファイル名>.elf <入力画像へのパス>
Cornertracker	xf::cornerTracker	./exe <入力ビデオ> <フレーム数> <Harris しきい値> <Harris コーナーをリセットするまでのフレーム数>
crop	xf::crop	./<実行ファイル名>.elf <入力画像へのパス>
Customconv	xf::filter2D	./<実行ファイル名>.elf <入力画像へのパス>
cvtColor IYUV2NV12	xf::iyuv2nv12	./<実行ファイル名>.elf <入力画像 1 へのパス> <入力画像 2 へのパス> <入力画像 3 へのパス>

表 11: xfOpenCV ライブラリ関数のハードウェアでの使用 (続き)

例	関数名	ハードウェアでの使用方法
cvtColor IYUV2RGBA	xf::iyuv2rgba	./<実行ファイル名>.elf <入力画像 1 へのパス> <入力画像 2 へのパス> <入力画像 3 へのパス>
cvtColor IYUV2YUV4	xf::iyuv2yuv4	./<実行ファイル名>.elf <入力画像 1 へのパス> <入力画像 2 へのパス> <入力画像 3 へのパス> <入力画像 4 へのパス> <入力画像 5 へのパス> <入力画像 6 へのパス>
cvtColor NV122IYUV	xf::nv122iyuv	./<実行ファイル名>.elf <入力画像 1 へのパス> <入力画像 2 へのパス>
cvtColor NV122RGBA	xf::nv122rgba	./<実行ファイル名>.elf <入力画像 1 へのパス> <入力画像 2 へのパス>
cvtColor NV122YUV4	xf::nv122yuv4	./<実行ファイル名>.elf <入力画像 1 へのパス> <入力画像 2 へのパス>
cvtColor NV212IYUV	xf::nv212iyuv	./<実行ファイル名>.elf <入力画像 1 へのパス> <入力画像 2 へのパス>
cvtColor NV212RGBA	xf::nv212rgba	./<実行ファイル名>.elf <入力画像 1 へのパス> <入力画像 2 へのパス>
cvtColor NV212YUV4	xf::nv212yuv4	./<実行ファイル名>.elf <入力画像 1 へのパス> <入力画像 2 へのパス>
cvtColor RGBA2YUV4	xf::rgba2yuv4	./<実行ファイル名>.elf <入力画像へのパス>
cvtColor RGBA2IYUV	xf::rgba2iyuv	./<実行ファイル名>.elf <入力画像へのパス>
cvtColor RGBA2NV12	xf::rgba2nv12	./<実行ファイル名>.elf <入力画像へのパス>
cvtColor RGBA2NV21	xf::rgba2nv21	./<実行ファイル名>.elf <入力画像へのパス>
cvtColor UYVY2IYUV	xf::uyvy2iyuv	./<実行ファイル名>.elf <入力画像へのパス>
cvtColor UYVY2NV12	xf::uyvy2nv12	./<実行ファイル名>.elf <入力画像へのパス>
cvtColor UYVY2RGBA	xf::uyvy2rgba	./<実行ファイル名>.elf <入力画像へのパス>
cvtColor YUYV2IYUV	xf::yuyv2iyuv	./<実行ファイル名>.elf <入力画像へのパス>
cvtColor YUYV2NV12	xf::yuyv2nv12	./<実行ファイル名>.elf <入力画像へのパス>
cvtColor YUYV2RGBA	xf::yuyv2rgba	./<実行ファイル名>.elf <入力画像へのパス>
Demosaicing	xf::demosaicing	./<実行ファイル名>.elf <入力画像へのパス>
Difference of Gaussian	xf::GaussianBlur、xf::duplicateMat、xf::delayMat、および xf::subtract	./<実行ファイル名>.elf <入力画像へのパス>
Dilation	xf::dilate	./<実行ファイル名>.elf <入力画像へのパス>
Erosion	xf::erode	./<実行ファイル名>.elf <入力画像へのパス>
Fast	xf::fast	./<実行ファイル名>.elf <入力画像へのパス>
Gaussianfilter	xf::GaussianBlur	./<実行ファイル名>.elf <入力画像へのパス>
Harris	xf::cornerHarris	./<実行ファイル名>.elf <入力画像へのパス>
Histogram	xf::calcHist	./<実行ファイル名>.elf <入力画像へのパス>
Histequalize	xf::equalizeHist	./<実行ファイル名>.elf <入力画像へのパス>
Hog	xf::HOGDescriptor	./<実行ファイル名>.elf <入力画像へのパス>
Houghlines	xf::HoughLines	./<実行ファイル名>.elf <入力画像へのパス>
inRange	xf::inRange	./<実行ファイル名>.elf <入力画像へのパス>
Integralimg	xf::integralImage	./<実行ファイル名>.elf <入力画像へのパス>
Lkdensepyrof	xf::densePyrOpticalFlow	./<実行ファイル名>.elf <入力画像 1 へのパス> <入力画像 2 へのパス>

表 11: xfOpenCV ライブラリ関数のハードウェアでの使用 (続き)

例	関数名	ハードウェアでの使用方法
Lknpyroflow	xf::DenseNonPyrLKOpticalFlow	./<実行ファイル名>.elf <入力画像 1 へのパス> <入力画像 2 へのパス>
Lut	xf::LUT	./<実行ファイル名>.elf <入力画像へのパス>
Kalman Filter	xf::KalmanFilter	./<実行ファイル名>.elf
Magnitude	xf::magnitude	./<実行ファイル名>.elf <入力画像へのパス>
Max	xf::Max	./<実行ファイル名>.elf <入力画像 1 へのパス> <入力画像 2 へのパス>
MaxS	xf::MaxS	./<実行ファイル名>.elf <入力画像へのパス>
meanshifttracking	xf::MeanShift	./<実行ファイル名>.elf <入力ビデオへのパス/入力画像ファイル> <追跡する物体の数>
meanstddev	xf::meanStdDev	./<実行ファイル名>.elf <入力画像へのパス>
medianblur	xf::medianBlur	./<実行ファイル名>.elf <入力画像へのパス>
Min	xf::Min	./<実行ファイル名>.elf <入力画像 1 へのパス> <入力画像 2 へのパス>
MinS	xf::MinS	./<実行ファイル名>.elf <入力画像へのパス>
Minmaxloc	xf::minMaxLoc	./<実行ファイル名>.elf <入力画像へのパス>
otsuthreshold	xf::OtsuThreshold	./<実行ファイル名>.elf <入力画像へのパス>
paintmask	xf::paintmask	./<実行ファイル名>.elf <入力画像へのパス>
Phase	xf::phase	./<実行ファイル名>.elf <入力画像へのパス>
Pyrdown	xf::pyrDown	./<実行ファイル名>.elf <入力画像へのパス>
Pyrup	xf::pyrUp	./<実行ファイル名>.elf <入力画像へのパス>
reduce	xf::reduce	./<実行ファイル名>.elf <入力画像へのパス>
remap	xf::remap	./<実行ファイル名>.elf <入力画像へのパス> <mapx データへのパス> <mapy データへのパス>
Resize	xf::resize	./<実行ファイル名>.elf <入力画像へのパス>
scale	xf::scale	./<実行ファイル名>.elf <入力画像へのパス>
scharrfilter	xf::Scharr	./<実行ファイル名>.elf <入力画像へのパス>
set	xf::set	./<実行ファイル名>.elf <入力画像へのパス>
SemiGlobalBM	xf::SemiGlobalBM	./<実行ファイル名>.elf <左画像へのパス> <右画像へのパス>
sobelfilter	xf::Sobel	./<実行ファイル名>.elf <入力画像へのパス>
stereopipeline	xf::StereoPipeline	./<実行ファイル名>.elf <左画像へのパス> <右画像へのパス>
stereolbm	xf::StereoBM	./<実行ファイル名>.elf <左画像へのパス> <右画像へのパス>
subRS	xf::SubRS	./<実行ファイル名>.elf <入力画像へのパス>
subS	xf::SubS	./<実行ファイル名>.elf <入力画像へのパス>
sum	xf::sum	./<実行ファイル名>.elf <入力画像 1 へのパス> <入力画像 2 へのパス>
Svm	xf::SVM	./<実行ファイル名>.elf
threshold	xf::Threshold	./<実行ファイル名>.elf <入力画像へのパス>
warptransform	xf::warpTransform	./<実行ファイル名>.elf <入力画像へのパス>

表 11: xfOpenCV ライブラリ関数のハードウェアでの使用 (続き)

例	関数名	ハードウェアでの使用方法
zero	xf::zero	./<実行ファイル名>.elf <入力画像へのパス>

# SDAccel での使用方法

この章では、SDAccel™ 環境での xfOpenCV の使用方法を説明します。カーネル、対応するホストコード、および SDAccel でサポートされるプラットフォーム用に xfOpenCV カーネルをコンパイルする makefile の作成方法を示し、カーネルをさまざまなエミュレーションモードおよびハードウェアで検証する方法を説明します。

## 使用要件

1. SDx™ 2019.1 以降のバージョンの有効なライセンスを入手してインストールします。
2. SDx で提供されるものと異なるライブラリを使用する場合は xfOpenCV ライブラリをインストールします。
3. SDx 2019.1 以降のバージョンでサポートされるプラットフォームのカードをインストールします。
4. ザイリンクス ランタイム (XRT) をインストールします。XRT では、ザイリンクス FPGA ヘソフトウェア インターフェイスが提供されます。
5. プラットフォームと共に libOpenCL.so をインストールする必要があります。

## SDAccel 設計手法

SDAccel™ を使用してカーネルをプラットフォームで機能させるには、次の 3 つのコンポーネントが必要です。

1. OpenCL コンストラクトを含むホストコード
2. HLS カーネルのラッパー
3. カーネルをエミュレーションまたはハードウェアでの実行用にコンパイルする makefile

## OpenCL を含むホストコード

ホストコードは、ホストで稼動するホストマシン用にコンパイルされ、ホストマシンに接続された FPGA を含むハードウェアにデータおよび制御信号を供給します。ホストコードは OpenCL コンストラクトを使用して記述されており、FPGA 上のカーネルを設定および実行する機能があります。次の関数は、ホストコードを使用して実行されます。

1. カーネルバイナリの FPGA への読み込み: `xcl::import_binary_file()` は、ビットストリームを読み込んで FPGA をプログラムし、必要なデータ処理を可能にします。
2. データ転送用にメモリバッファを設定: データをハードウェア上の DDR メモリとデータを送受信する必要があります。`cl::Buffers` は、ハードウェアとのデータ転送用に必要なメモリを割り当てるために作成されます。

- ハードウェアとのデータ転送: enqueueWriteBuffer() および enqueueReadBuffer() は、必要な時にハードウェアとデータを転送するために使用されます。
- FPGA デバイスでのカーネルの実行: FPGA でカーネルを実行する関数があります。1つのカーネル実行または複数のカーネル実行(カーネルどうしは同期または非同期)があります。よく使用されるコマンドは enqueueTask() です。
- カーネル実行のパフォーマンスのプロファイリング: OpenCL で記述されたホスト コードにより、FPGA 上でのカーネルの実行時間を計測できます。ザイリンクスの例でプロファイリングに使用される関数は getProfilingInfo() です。

## HLS カーネルのラッパー

すべての xfOpenCV カーネルには、C++ 関数テンプレート (<Github repo>/include に配置) と、xf::Mat クラスのオブジェクトとして画像コンテナが提供されています。また、これらのカーネルは、ストリームベース(画像全体を連続して読み出し)またはメモリ マップド(画像データにブロック単位でアクセス)の両方で機能します。

SDAccel フロー(OpenCL)では、カーネルインターフェイスは幅が2のべき数のメモリ ポインターである必要があります。そのため、xfOpenCV カーネルとデータを転送するには、メモリ ポインターを xf::Mat クラスのデータ型に変換(およびその逆変換)するためのグルー ロジックが必要です。カーネルとこのグルー ロジックを含むラッパーがビルドされます。次の例では、異なるカーネル(<Github repo>/include にある xfOpenCV カーネル)のタイプ(ストリームおよびメモリ マップド)を処理する方法を示します。

## ストリーム ベース カーネル

ポインターから xf::Mat およびその逆の変換を実行するため、xfOpenCV の一部として2つのアダプター関数 xf::Array2xfMat() および xf::xfMat2Array() が含まれます。これらの xf::Mat オブジェクトは、HLS プラグマを使用して2以上の深さでストリームとして呼び出す必要があります。カーネルの最上位(ラッパー)関数は、次のようになります。

```
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);
}
}
```

上記では、xf::Mat のデータがストリーミング入力/出力されると想定されています。1つの xfOpenCV 関数の代わりに、複数の関数のパイプラインを作成することもできます。

異なるサイズの異なる入力を使用するストリーム ベース カーネルでは、アダプター関数の複数のインスタンスが必要です。

```
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, in_mat3, out_mat...);
xf::xfMat2Array<...> (gmem_out, out_mat);
}
```

ストリームベースのインプリメンテーションでは、特定の構成の xfcv カーネルでの必要に応じて、入力 AXI からデータをフェッチし、xfMat にプッシュする必要があります。同様に、同じ操作を xfcv カーネルの出力に対しても実行する必要があります。このため、xf::Array2xfMat() と xf::xfMat2Array() の 2 つのユーティリティ関数が提供されています。

## Array2xfMat

入力配列を xf::Mat に変換します。xfOpenCV カーネルでは、入力のデータ型は xf::Mat である必要があります。この関数は、配列ポインターから読み出し、xf::Mat が作成された設定 (ビット深さ、チャンネル数、ピクセルの並列処理) に基づいて xf::Mat に書き込みます。

```
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)
```

表 12: Array2xfMat 関数のパラメーターの説明

パラメーター	説明
PTR_WIDTH	入力ポインターのデータ幅。有効な値は 8 ～ 512 の 2 のべき乗です。
MAT_T	入力 Mat 型。XF_8UC1、XF_16UC1、XF_8UC3、XF_8UC4 など。
ROWS	画像の最大高さ
COLS	画像の最大幅
NPC	並列計算されるピクセルの数。XF_NPPC1、XF_NPPC8 など。
srcPtr	入力ポインター。PTR_WIDTH に基づくポインターのタイプ。
dstMat	xf::Mat 型の出力画像

## xfMat2Array

入力 xf::Mat を出力配列に変換します。xf::kernel 関数の出力は xf::Mat で、これを出力ポインターに変換する必要があります。

```
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)
```

表 13: xfMat2Array 関数のパラメーターの説明

パラメーター	説明
PTR_WIDTH	出力ポインターのデータ幅。8 ～ 512 の 2 のべき数で指定。

表 13: xfMat2Array 関数のパラメーターの説明 (続き)

パラメーター	説明
MAT_T	入力 Mat 型。XF_8UC1、XF_16UC1、XF_8UC3、XF_8UC4 など。
ROWS	画像の最大高さ
COLS	画像の最大幅
NPC	並列計算されるピクセルの数。XF_NPPC1、XF_NPPC8 など。
dstPtr	出力ポインター。PTR_WIDTH に基づくポインターのタイプ。
srcMat	xf::Mat 型の入力画像

インターフェイス ポインターの幅

次の表に、異なる構成でのポインターの最小幅を示します。

表 14: 異なる Mat 型の最小および最大ポインター幅

Mat 型	並列処理	最小 PTR_WIDTH	最大 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

## メモリ マップド カーネル

切り抜き、平均値シフト追跡、境界ボックスなどのメモリ マップ ベースのカーネルでは、入力読み出しはアルゴリズムの要件に基づく特定のメモリ ブロックです。ストリーミング インターフェイスでは画像をラスタ走査方式で読み出す必要がありますが、メモリ マップド カーネルでは次の手法で処理されます。

```
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...);
}
}
```

オブジェクトの作成時に gmem ポインターを xf::Mat オブジェクトにマップし、インターフェイスでこれらの xf::Mat オブジェクトを使用してメモリ マップド カーネルを呼び出します。ポインターのサイズは xf::xfopencv-func に必要なサイズと同じにする必要があります。ストリーミングでは、これより大きいポインター サイズ(512 ビットまで)がサポートされます。



## makefile

現在の使用モデルでは、SDAccel の xfOpenCV を使用してアプリケーションをビルドするのに makefile ベースのフローのみが提供されています。makefile の例は、GitHub のサンプル セクションから入手できます。

## SDAccel のライブラリを使用したデザイン例

次の例は複数のカーネルの例で、異なるカーネルをパイプラインで順次実行するアプリケーションを示します。この例では、Canny 法によるエッジ検出が実行され、2つのカーネルが Canny およびエッジ検出に使用されます。Canny 関数はグレースケール画像を入力として取り込み、エッジ情報が3つの状態(弱いエッジ(1)、強いエッジ(3)、および背景(0))でエッジ検出に供給され、弱いエッジが取り除かれます。最初の処理はストリーミングベースのインプリメンテーションで、後の処理はメモリマップで実行されます。

### ホスト コード

次に、Canny エッジ検出例のホストコードを示します。このホストコードは、必要なデータを処理する FPGA を含む OpenCL プラットフォームを設定します。xfOpenCV の場合、データは画像です。画像の読み出しおよび書き込みは、xfOpenCV の関数への呼び出しを使用して実行します。

```
// 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 startededge= 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,&startededge);
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();
```

## 最上位カーネル

次に、必要なすべてのグルー ロジックを含む最上位/ラッパー関数を示します。

```
// 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_threshol
d);

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);

}
}

```

## 機能の評価

カーネルをビルドして機能の評価するには、ソフトウェアエミュレーション、ハードウェアエミュレーション、FPGAを含むサポートされるハードウェア上での直接実行を使用できます。PCIe ベース プラットフォームでは、次のコマンドを実行して環境を設定します。

```
$ 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
```

## ソフトウェアエミュレーション

ソフトウェアエミュレーションは、カーネルのCシミュレーションを実行するのと同様です。コンパイルにかかる時間は最小限なので、カーネルのテストの第一段階としてお勧めします。ソフトウェアエミュレーションを実行する手順は、次のとおりです。

```
$ 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/tools7/opencv:7usr/lib64
$ ./<executable> <args>
```

## ハードウェアエミュレーション

ハードウェアエミュレーションでは、C/C++ コードの合成後に生成された RTL に対してテストを実行します。このシミュレーションは RTL 上で実行されるので、ソフトウェアエミュレーションよりも時間がかかります。ハードウェアエミュレーションを実行する手順は、次のとおりです。

```
$ 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/tools7/opencv:7usr/lib64
$ ./<executable> <args>
```

## ハードウェア上でのテスト

ハードウェア上でテストするには、カーネルをビットストリームにコンパイル(ハードウェア用にビルド)する必要があります。

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

これには、C/C++ コードを RTL に変換し、合成およびインプリメンテーションを実行してビットストリームを生成する必要があるため、時間がかかります。例をビルドした対応する DSA 用のドライバーをインストールする必要があります。次に、ハードウェア上でカーネルを実行する手順を示します。

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

## HLS の使用

xfOpenCV ライブラリを使用して、Vivado® HLS でアプリケーションをビルドできます。このセクションでは、Vivado HLS 2019.1 環境のデザインに xfOpenCV ライブラリ コンポーネントを統合する方法を説明します。C シミュレーション、C 合成、C/RTL 協調シミュレーション、および RTL を IP としてエクスポートなど、Vivado HLS 2019.1 使用フローで 1 つのライブラリ コンポーネントを実行する手順を示します。

Vivado HLS 2019.1 で使用モデルが正しく機能するようにするため、次の変更を加える必要があります。

1. 適切なコンパイル時間オプションの使用: HLS で xfOpenCV 関数を使用する際、コンパイル時に `-D__SDSVHLS__` および `-std=c++0x` オプションを使用する必要があります。
2. インターフェイス レベル引数へのインターフェイス プラグマの指定: 最上位インターフェイス引数を (1 つ以上の読み出し/書き込みアクセスを持つ) ポインターとして含む関数には、`m_axi` インターフェイス プラグマを指定する必要があります。次に例を示します。

```
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 スタンドアロン モード

HLS スタンドアロン モードは、次の 2 つのモードで使用できます。

1. Tcl スクリプト モード
2. GUI モード

### Tcl スクリプト モード

Tcl スクリプトを使用して HLS スタンドアロン モードを操作するには、次の手順に従います。

1. Vivado® HLS Tcl スクリプト ファイルで、すべての `add_files` セクションの `cflags` をアップデートします。
2. `xfOpenCV/include` ディレクトリへのパスを追加します。このディレクトリには、ライブラリに必要なヘッダー ファイルがすべて含まれます。
3. `-D__SDSVHLS__` および `-std=c++0x` コンパイラ フラグを追加します。

**注記:** Windows で Vivado HLS を使用する場合は、C シミュレーションおよび協調シミュレーションにのみ `-std=c++0x` フラグを追加してください。合成を実行する場合は、このフラグは含めないでください。

例:

ソース ファイルのフラグ設定:

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

テストベンチ ファイルのフラグ設定:

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

## GUI モード

GUI を使用して HLS スタンドアロン モードを操作するには、次の手順に従います。

1. Vivado® HLS を GUI モードで開いて新規プロジェクトを作成します。
2. プロジェクト名を指定します (「Dilation」など)。
3. [Browse] をクリックし、プロジェクトを保存するワークスペース フォルダを指定します。
4. [Next] をクリックします。
5. [Design Files] セクションに、examples フォルダにある accel.cpp ファイルを追加します。[Top Function] に最上位関数名 (ここでは dilation\_accel) を入力します。
6. [Next] をクリックします。
7. [TestBench Files] セクションに、tb.cpp ファイルを追加します。
8. [Next] をクリックします。
9. クロック周期 (ここでは 10 ns) を指定します。
10. パーツを選択します (例 xczu9eg-ffvb1156-2-i)。
11. [Finish] をクリックします。
12. 作成したプロジェクトを右クリックし、[Project Settings] をクリックします。
13. 開いたタブで [Simulation] を選択します。
14. [TestBench Files] セクションで追加したファイルが表示されます。ファイルを選択して [Edit CFLAGS] をクリックします。
15. 「-I<path-to-include-directory> -D\_\_SDSVHLS\_\_ -std=c++0x」と入力します。  
**注記:** Windows で Vivado HLS を使用する場合は、C シミュレーションおよび協調シミュレーションにのみ -std=c++0x フラグを追加してください。合成を実行する場合は、このフラグは含めないでください。
16. [Synthesis] を選択し、表示されているすべてのファイルに対して上記の手順を繰り返します。
17. [OK] をクリックします。
18. C シミュレーションを実行し、[Clean Build] を選択して必要な入力引数を指定します。
19. [OK] をクリックします。
20. 生成された出力ファイル/画像はすべて solution1->csim->build にあります。
21. C Synthesis を実行します。
22. 適切な入力引数を指定して Co-simulation を実行します。
23. 協調シミュレーションのステータスはコンソールで確認できます。

## 協調シミュレーション用の制約

xfOpenCV 関数の協調シミュレーションを実行するには、いくつかの制限があります。これらは次のとおりです。

1. 複数のアクセラレータを含む関数はサポートされていません。
2. コンパイラおよびシミュレータ (gcc、XSim) では、HLS がデフォルトになります。
3. HLS では複数のカーネルの統合がサポートされないため、現在のフローでも複数のカーネルの統合はサポートされません。そのため、ピラミッド型オプティカル フロー関数および Canny 法によるエッジ検出関数はこのフローではサポートされません。
4. config.h ファイルで設定されている最大画像サイズ (HEIGHT および WIDTH) は、実際の入力画像サイズである必要があります。

## AXI ビデオ インターフェイス関数

xfOpenCV には、xf::Mat をザイリンクス ビデオ ストリーミング インターフェイスに、およびその逆に変換する関数が含まれます。xf::AXIvideo2xfMat() および xf::xfMat2AXIvideo() は、Vivado® IP インテグレーターの xfOpenCV 関数の IP に対するビデオ インターフェイスとして機能します。cvMat2AXIvideoxf <NPC> および AXIvideo2cvMatxf <NPC> は、ホスト側で使用されます。

表 15: AXI ビデオ インターフェイス関数

ビデオ ライブラリ関数	説明
AXIvideo2xfMat	AXI4 ビデオ ストリームのデータを xf::Mat フォーマットに変換します。
xfMat2AXIvideo	xf::Mat フォーマットで格納されたデータを AXI4 ビデオ ストリームに変換します。
cvMat2AXIvideoxf	cv::Mat フォーマットで格納されたデータを AXI4 ビデオ ストリームに変換します。
AXIvideo2cvMatxf	AXI4 ビデオ ストリームのデータを cv::Mat フォーマットに変換します。

## AXIvideo2xfMat

AXIvideo2xfMat 関数は、AXI4 Streaming Video を使用して画像のシーケンスを受信し、xf::Mat フォーマットの画像を生成します。

### API 構文

```
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)
```

### パラメーターの説明

次の表に、テンプレートと関数のパラメーターを説明します。



表 16: AXIvideo2cvMatxf 関数のパラメーターの説明

パラメーター	説明
W	AXI4-Stream のデータ幅。推奨される値はピクセル深さ。
T	画像のピクセルのデータ型。1 チャンネル (XF_8UC1)。ピクセルのデータ幅は、W 以下である必要があります。
ROWS	入力画像の最大高さ。
COLS	入力画像の最大幅。
NPC	1 サイクルごとに処理されるピクセル数。使用可能なオプションは、1 ピクセルの場合は XF_NPPC1、8 ピクセルの場合は XF_NPPC8。
AXI_video_strm	ap_axiu (axi protocol) 型の HLS ストリーム。
img	入力画像。

TLAST 入力を検出し、予期しないライン長を示す ERROR\_IO\_EOL\_EARLY または ERROR\_IO\_EOL\_LATE のビット エラーを返します。

AXI インターフェイスの詳細は、『AXI リファレンス ガイド』 (UG761) を参照してください。

## xfMat2AXIvideo

Mat2AXI ビデオ関数は、cv::Mat フォーマットの画像のシーケンスを受信し、AXI4-Stream ビデオ プロトコルを使用してエンコードします。

### API 構文

```
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)
```

### パラメーターの説明

次の表に、テンプレートと関数のパラメーターを説明します。

表 17: xfMat2AXIvideo 関数のパラメーターの説明

パラメーター	説明
W	AXI4-Stream のデータ幅。推奨される値はピクセル深さ。
T	画像のピクセルのデータ型。1 チャンネル (XF_8UC1)。ピクセルのデータ幅は、W 以下である必要があります。
ROWS	入力画像の最大高さ。
COLS	入力画像の最大幅。
NPC	1 サイクルごとに処理されるピクセル数。使用可能なオプションは、1 ピクセルの場合は XF_NPPC1、8 ピクセルの場合は XF_NPPC8。
AXI_video_strm	ap_axiu (axi protocol) 型の HLS ストリーム。
img	出力画像。

この関数は値 0 を返します。

**注記:** データフローのすべての関数の NPC 値は、同じ値である必要があります。同じでない値があると、HLS でコンパイルエラーが生成されます。

## cvMat2AXIvideoxf

cvMat2Axivideoxf 関数は、cv::Mat フォーマットの画像を受信し、AXI4 ストリーミング ビデオ画像を生成します。

### API 構文

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

### パラメーターの説明

次の表に、テンプレートと関数のパラメーターを説明します。

表 18: AXIvideo2cvMatxf 関数のパラメーターの説明

パラメーター	説明
W	AXI4-Stream のデータ幅。推奨される値はピクセル深さ。
NPC	1 サイクルごとに処理されるピクセル数。使用可能なオプションは、1 ピクセルの場合は XF_NPPC1、8 ピクセルの場合は XF_NPPC8。
AXI_video_strm	ap_axiu (axi protocol) 型の HLS ストリーム。
cv_mat	入力画像。

## AXIvideo2cvMatxf

Axivideo2cvMatxf 関数は、画像を AXI4 ストリーミング ビデオとして受信し、cv::Mat フォーマットの画像を生成します。

### API 構文

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

### パラメーターの説明

次の表に、テンプレートと関数のパラメーターを説明します。

表 19: AXIvideo2cvMatxf 関数のパラメーターの説明

パラメーター	説明
W	AXI4-Stream のデータ幅。推奨される値はピクセル深さ。
NPC	1 サイクルごとに処理されるピクセル数。使用可能なオプションは、1 ピクセルの場合は XF_NPPC1、8 ピクセルの場合は XF_NPPC8。
AXI_video_strm	ap_axiu (axi protocol) 型の HLS ストリーム。

表 19: AXIvideo2cvMatxf 関数のパラメーターの説明 (続き)

パラメーター	説明
cv_mat	出力画像。

# xfOpenCV ライブラリ API リファレンス

FPGA デバイス上のローカル メモリ割り当てを円滑にするため、xfOpenCV ライブラリ関数はコンパイル時パラメータを持つテンプレートとして提供されています。データは `cv::Mat` から `xf::Mat` にコピーされ、できるだけ高いパフォーマンスを達成するため物理的に隣接したメモリに保存されます。処理後、`xf::Mat` の出力が `cv::Mat` にコピーされ、メモリに書き込まれます。

## xf::Mat 画像コンテナ クラス

`xf::Mat` は、画像データとその属性を格納するコンテナとして動作するテンプレートクラスです。

**注記:** `xf::Mat` 画像コンテナ クラスは OpenCV ライブラリの `cv::Mat` クラスと類似しています。

### クラス定義

```
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
        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);
};

```

## パラメーターの説明

次の表に、`xf::Mat` クラスのパラメーターと説明を示します。

表 20: `xf::Mat` クラスのパラメーターと説明

パラメーター	説明
<code>rows</code>	画像内の行数または画像の高さ。
<code>cols</code>	画像内の列数または画像の幅。
<code>size</code>	データ メンバーに格納されるワード数。値は <code>rows*cols/(number of pixels packed per word)</code> を使用して算出されます。
<code>allocatedFlag</code>	メモリ割り当てのステータスを示すフラグ。
<code>*data</code>	画像のピクセルを格納するワードへのクラス パラメーターおよびポインター。

## メンバー関数の説明

次の表に、メンバー関数とその説明を示します。

表 21: `xf::Mat` メンバー関数の説明

メンバー関数	説明
<code>Mat()</code>	デフォルトのコンストラクター。ROWS および COLS テンプレート パラメーターを使用して <code>Mat</code> オブジェクトのサイズを初期化します。
<code>Mat(int _rows, int _cols)</code>	<code>_rows</code> および <code>_cols</code> 引数を使用して <code>Mat</code> オブジェクトを初期化します。
<code>Mat(const xf::Mat &amp;_src)</code>	<code>Mat</code> オブジェクトを別の <code>Mat</code> オブジェクトにクローンするのに役立つコンストラクター。新しく作成されるコンストラクター用に、新しいメモリが割り当てられます。
<code>Mat(int _rows, int _cols, void *_data)</code>	<code>_rows</code> 、 <code>_cols</code> 、および <code>_data</code> 引数を使用して <code>Mat</code> オブジェクトを初期化します。このコンストラクターを使用すると、 <code>Mat</code> オブジェクトの <code>*data</code> メンバーで <code>_data</code> 引数に割り当てられたメモリが指定されます。 <code>*data</code> メンバー用に新しいメモリが割り当てられることはありません。
<code>convertTo(Mat&lt;DST_T,ROWS, COLS, NPC&gt; &amp;dst, int otype, double alpha=1, double beta=0)</code>	<a href="#">xf::convertTo</a> を参照
<code>copyTo(* fromData)</code>	データ ポインターからのデータをコンストラクター内で割り当てられた物理的に隣接したメモリにコピーします。
<code>copyFrom()</code>	ポインターを <code>*data</code> メンバーの最初の位置に戻します。
<code>read(int index)</code>	指定の場所から値を読み出し、複数ピクセル/クロックのパケット値として返します。
<code>read_float(int index)</code>	指定の場所から値を読み出し、浮動小数点値として返します。
<code>write(int index, XF_TNAME(T,NPC) val)</code>	複数ピクセル/クロックのパケット値を指定の場所書き込みます。
<code>write_float(int index, float val)</code>	浮動小数点値を指定の場所書き込みます。
<code>type()</code>	画像のデータ型を返します。
<code>depth()</code>	画像の深さを返します。
<code>channels()</code>	画像のチャンネル数を返します。
<code>~Mat()</code>	<code>Mat</code> オブジェクトのデフォルト デストラクターです。

## テンプレートのパラメーターの説明

`xf::Mat` クラスのテンプレート パラメーターは、ピクセルの深さ、画像のチャンネル数、各ワードにパックされるピクセル数、画像の行および列の最大数を設定するのに使用します。次の表に、テンプレート パラメーターとその説明を示します。

表 22: `xf::Mat` テンプレート パラメーターの説明

パラメーター	説明
TYPE	ピクセルのデータ型。たとえば、XF_8UC1 は 8 ビットの符号なしの 1 チャンネル ピクセルを意味します。その他のデータ型は <code>include/common/xf_params.h</code> を参照してください。
HEIGHT	画像の最大高さ。
WIDTH	画像の最大幅。
NPC	各ワードにパックされるピクセル数。たとえば、XF_NPPC1 は毎ワード 1 ピクセル、XF_NPPC8 は毎ワード 8 ピクセルを意味します。

## ピクセル レベルの並列処理

xfOpenCV から関数にインプリメントされる並列処理の量は、設定可能なパラメーターです。ほとんどの関数では、2 つのデータ処理オプションがあります。

- 1 ピクセル処理
- 8 ピクセルの並列処理

次の表に、特定の関数で必要な並列処理のレベルを指定するオプションを示します。

表 23: 並列処理のレベルを指定するのに使用できるオプション

オプション	説明
XF_NPPC1	1 クロック サイクルごとに 1 ピクセルを処理
XF_NPPC2	1 クロック サイクルごとに 2 ピクセルを処理
XF_NPPC4	1 クロック サイクルごとに 4 ピクセルを処理
XF_NPPC8	1 クロック サイクルごとに 8 ピクセルを処理

## 並列処理に使用するマクロ

次の 2 つのマクロは、並列処理で使用できるように定義されています。

- `XF_NPIXPERCYCLE(flags)` マクロ: 1 サイクルごとに処理するピクセル数を返します。
  - `XF_NPIXPERCYCLE(XF_NPPC1)` は 1 を返します。
  - `XF_NPIXPERCYCLE(XF_NPPC2)` は 2 を返します。
  - `XF_NPIXPERCYCLE(XF_NPPC4)` は 4 を返します。
  - `XF_NPIXPERCYCLE(XF_NPPC8)` は 8 を返します。
- `XF_BITSHIFT(flags)` マクロ: 並列処理で最終画像データ画像サイズを得るために画像サイズを右にシフトする回数を返します。
  - `XF_BITSHIFT(XF_NPPC1)` は 0 を返します。
  - `XF_BITSHIFT(XF_NPPC2)` は 1 を返します。

- `XF_BITSHIFT(XF_NPPC4)` は 2 を返します。
- `XF_BITSHIFT(XF_NPPC8)` は 3 を返します。

### ピクセルのデータ型

パラメーターのデータ型は、画像内のピクセルの深さとチャンネル数の組み合わせによって異なります。このパラメーターの一般的な命名方法は、次のとおりです。

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

たとえば、8 ビット ピクセル、符号なし、1 チャンネルのデータ型は `XF_8UC1` です。

次の表に、`xf::Mat` クラスの使用可能なデータ型を示します。

表 24: `xf::Mat` クラス - 使用可能なデータ型

オプション	1 ピクセルごとのビット数	符号なし/符号付き/浮動小数点型	チャンネル数
<code>XF_8UC1</code>	8	符号なし	1
<code>XF_16UC1</code>	16	符号なし	1
<code>XF_16SC1</code>	16	符号付き	1
<code>XF_32UC1</code>	32	符号なし	1
<code>XF_32FC1</code>	32	浮動小数点	1
<code>XF_32SC1</code>	32	符号付き	1
<code>XF_8UC2</code>	8	符号なし	2
<code>XF_8UC4</code>	8	符号なし	4
<code>XF_8UC3</code>	8	符号なし	3
<code>XF_2UC1</code>	2	符号なし	1

### データ型の操作

1 クロック サイクルごとに処理するピクセル数と型パラメーターに基づき、異なるデータ型を使用できます。xfOpenCV ライブラリでは、これらのデータ型が内部処理および `xf::Mat` クラス内で使用されます。次に、サポートされるデータ型をいくつか示します。

- `XF_TNAME(TYPE, NPPC):xf::Mat` オブジェクトのデータ メンバーのデータ型を返します。たとえば、`XF_TNAME(XF_8UC1, XF_NPPC8)` は `ap_uint<64>` を返します。
- `Word width = pixel depth * number of channels * number of pixels to process per cycle (NPPC)`
- `XF_DTUNAME(TYPE, NPPC):` ピクセルのデータ型を返します。たとえば、`XF_DTUNAME(XF_32FC1, XF_NPPC1)` は `float` を返します。
- `XF_PTSNAME(TYPE, NPPC):` ピクセルの C データ型を返します。たとえば、`XF_PTSNAME(XF_16UC1, XF_NPPC2)` は `unsigned short` を返します。

**注記:** `ap_uint<>`、`ap_int<>`、`ap_fixed<>`、および `ap_ufixed<>` 型は、高位合成 (HLS) ライブラリに含まれます。詳細は、『Vivado Design Suite ユーザー ガイド: 高位合成』(UG902) を参照してください。



## コード例

次のコードは、Zynq® UltraScale™ プラットフォーム用に SDSoC™ ツールを使用して、画像にガウシアン フィルターを構築するのに必要な設定を示しています。

**注記:** ビデオがストリーム入力されるリアルタイム アプリケーションの場合、フレームバッファの位置が `xf::Mat` でライブラリ関数を使用して処理されるようにしてください。結果の位置ポインターが渡されて IP が表示されるようになります。

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
#else
#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
}
```

データがSDSoC データムーバーを使用して外部メモリからフェッチされて、設定モードによって8ビットまたは64ビットパケットの関数に転送されます。1ピクセルごとに8ビットの場合、8ピクセルを64ビットにパックできます。つまり、8ピクセルを並列処理に使用できます。

xf\_config\_params.h ファイルで FILTER\_SIZE\_3 と NO マクロをイネーブルにします。マクロでフィルターサイズが3x3に設定され、#define NO 1 マクロで1ピクセルの並列処理がイネーブルになります。

xf\_gaussian\_filter.hpp ファイルで SDSoC 特有のプラグマを指定します。

```
#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"])
```

**注記:** SDSoC のハードウェアアクセラレータ関数で使用するプラグマについては、『SDSoC 環境ユーザーガイド』(UG1027)を参照してください。

## ソフトウェア用の追加ユーティリティ関数

### xf::imread

関数 xf::imread は、指定のファイルパスから画像を読み込み、xf::Mat にコピーして戻します。画像を読み込むことができない場合は(ファイルがない、権限が不適切、サポートされていない無効なフォーマットなど)、関数は終了し、0以外の戻りコードとエラー文が返されます。

**注記:** 協調シミュレーションなどの HLS スタンドアロン モードでは、xf::imread ではなく、cv::imread を使用してその後に copyTo 関数を使用します。

#### API 構文

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

#### パラメーターの説明

次の表に、テンプレートと関数のパラメーターを説明します。

表 25: xf::imread 関数のパラメーターの説明

パラメーター	説明
PTYPE	入力ピクセルのデータ型。値は type 引数の値によります。
ROWS	読み込む画像の最大高さ
COLS	読み込む画像の最大幅
NPC	1 サイクルごとに処理されるピクセル数。1 ピクセルの場合は XF_NPPC1、8 ピクセルの場合は XF_NPPC8。
filename	読み込むファイルの名前
type	画像のタイプを示すフラグ。値は次のとおりです。 <ul style="list-style-type: none"> <li>0: グレー スケール</li> <li>1: カラー画像</li> </ul>

## xf::imwrite

関数 xf::imwrite は、画像を指定の xf::Mat からのファイルに保存します。画像のフォーマットは、ファイル名の拡張子に基づいて選択されます。この関数は内部で cv::imwrite を使用するので、cv::imwrite の制限がすべて xf::imwrite にも適用されます。

### API 構文

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

### パラメーターの説明

次の表に、テンプレートと関数のパラメーターを説明します。

表 26: xf::imwrite 関数のパラメーターの説明

パラメーター	説明
PTYPE	入力ピクセルのデータ型。XF_8UC1、XF_16UC1、XF_8UC4、および XF_16UC4 をサポート。
ROWS	読み込む画像の最大高さ
COLS	読み込む画像の最大幅
NPC	1 サイクルごとに処理されるピクセル数。1 ピクセルの場合は XF_NPPC1、8 ピクセルの場合は XF_NPPC8。
img_name	拡張子を含むファイルの名前
img	保存する xf::Mat 配列

## xf::absDiff

関数 xf::absDiff は、xf::Mat と cv::Mat の各ピクセルの絶対差分を計算し、cv::Mat の差分値を返します。

## API 構文

```
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 )
```

## パラメーターの説明

次の表に、テンプレートと関数のパラメーターを説明します。

表 27: xf::absDiff 関数のパラメーターの説明

パラメーター	説明
PTYPE	入力ピクセルのデータ型
ROWS	読み込む画像の最大高さ
COLS	読み込む画像の最大幅
NPC	1 サイクルごとに処理されるピクセル数。1 ピクセル動作の場合は XF_NPPC1、4 ピクセル動作の場合は XF_NPPC4、8 ピクセル動作の場合は XF_NPPC8。
cv_img	比較する cv::Mat 配列
xf_img	比較する xf::Mat 配列
diff_img	出力差分画像 (cv::Mat)

## xf::convertTo

xf::convertTo 関数は、入力画像の各ピクセルに対してビット深さ変換を実行します。ソース ピクセル値を適切な型変換を使用してターゲット データ型に変換します。

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

**注記:** 出力 Mat と入力 Mat は異なっている必要があります。つまり、変換された画像を入力画像の Mat に格納することはできません。

## API 構文

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

## パラメーターの説明

次の表に、テンプレートと関数のパラメーターを説明します。

表 28: xf::convertTo 関数のパラメーターの説明

パラメーター	説明
DST_T	出力ピクセルのデータ型。有効な値は XF_8UC1、XF_16UC1、XF_16SC1、および XF_32SC1 です。
ROWS	読み込む画像の最大高さ
COLS	読み込む画像の最大幅
NPC	1 サイクルごとに処理されるピクセル数。1 ピクセル動作の場合は XF_NPPC1、4 ピクセル動作の場合は XF_NPPC4、8 ピクセル動作の場合は XF_NPPC8。XF_32SC1 と XF_NPPC8 の組み合わせはサポートされません。

表 28: xf::convertTo 関数のパラメーターの説明 (続き)

パラメーター	説明
dst	変換後の xF Mat
ctype	変換タイプ。有効な値は次のとおりです。 //Down-convert: <ul style="list-style-type: none"> <li>XF_CONVERT_16U_TO_8U</li> <li>XF_CONVERT_16S_TO_8U</li> <li>XF_CONVERT_32S_TO_8U</li> <li>XF_CONVERT_32S_TO_16U</li> <li>XF_CONVERT_32S_TO_16S</li> </ul> //Up-convert: <ul style="list-style-type: none"> <li>XF_CONVERT_8U_TO_16U</li> <li>XF_CONVERT_8U_TO_16S</li> <li>XF_CONVERT_8U_TO_32S</li> <li>XF_CONVERT_16U_TO_32S</li> <li>XF_CONVERT_16S_TO_32S</li> </ul>
alpha	オプションのスケール係数
beta	スケール値に追加するデルタ (オプション)

## xfOpenCV ライブラリ関数

xfOpenCV ライブラリは、Zynq-7000 および Zynq UltraScale+ MPSoC デバイス用に最適化された OpenCV 関数のセットです。次の表に、xfOpenCV ライブラリ関数を示します。

表 29: xfOpenCV ライブラリ関数

計算	入力処理	フィルター	その他
絶対差分 (absdiff)	ビット深度変換 (convertTo)	バイラテラル フィルター (bilateralFilter)	Canny 法によるエッジ検出 (Canny)
累積 (accumulate)	チャンネル結合 (merge)	ボックス フィルター (boxFilter)	FAST コーナー検出 (fast)
累積二乗 (accumulateSquare)	チャンネル抽出 (extractChannel)	カスタムたたみ込み (filter2D)	Harris コーナー検出 (cornerHarris)
累積重み (accumulateWeighted)	色変換	膨張 (dilate)	ヒストグラム計算 (calcHist)
Atan2	ヒストグラム均一化 (equalizeHist)	収縮 (erode)	高密度ピラミッド型 LK オプティカル フロー (DensePyrOpticalFlow)

表 29: xfOpenCV ライブラリ関数 (続き)

計算	入力処理	フィルター	その他
ビット単位 AND (bitwise_and)、ビット単位 NOT (bitwise_not)、ビット単位 OR (bitwise_or)、ビット単位 XOR (bitwise_xor)	ルックアップテーブル (LUT)	ガウシアン フィルター (GaussianBlur)	高密度非ピラミッド型 LK オプティカル フロー (DenseNonPyrLKOpticalFlow)
勾配の大きさ (magnitude)	リマップ (remap)	Sobel フィルター (Sobel)	MinMax ロケーション (minMaxLoc)
勾配位相 (phase)	解像度変換/リサイズ (resize)	メジアン フィルター (medianBlur)	しきい値処理 (Threshold)
積分画像 (integral)	convertScaleAbs	Scharr フィルター (Scharr)	SVM
インバース/逆数 (Inverse)	モザイク処理 (demosaicing)		Otsu 法のしきい値処理 (OtsuThreshold)
ピクセル加算 (add)	切り抜き (crop)		平均値シフト追跡 (MeanShift)
ピクセル乗算 (multiply)	Reduce		HOG (HOGDescriptor)
ピクセル減算 (subtract)	境界ボックス (BoundingBox)		ステレオ ローカル ブロック マッチング (StereoBM)
平方根 (Sqrt)			ワープ変換 (warpTransform)
平均および標準偏差 (meanStdDev)			ピラミッド アップ (pyrUp)
スカラー値との加算 (addS)、比較 (compare)、スカラー値との比較 (compareS)、最大 (Max)、MaxS、最小 (Min)、MinS、設定 (set)、subRS、SubS、zero			ピラミッド ダウン (pyrDown)
和 (sum)			遅延 (delayMat)
加重和 (addWeighted)			複製 (duplicateMat)
			色しきい値処理 (colorthresholding)
			BGR2HSV
			InitUndistortRectifyMapInverse
			ハフ変換に基づく線分の抽出 (Houghlines)
			ステレオ視差推定用のセミグローバル法 (SemiGlobalBM)
			paintmask
			inRange
			カルマン フィルター

**注記:**

1. HoughLines および HOG (RB モード) を除き、関数でサポートされる最大解像度は 4K です。

**注記:** Zynq-7000 SoC ZC702 デバイスでは、1 サイクルごとに 8 ピクセル モードの解像度変換/リサイズ (resize)、高密度ピラミッド型 LK オプティカル フロー (DensePyrOpticalFlow)、高密度非ピラミッド型 LK オプティカル フロー (DenseNonPyrLKOpticalFlow) 関数はリソース使用量が多くなるためサポートされません。

**注記:** 1 クロックごとのピクセル数は、デバイスでサポート可能な最大バス幅によって異なります。

たとえば、Zynq-7000 SoC には 64 ビット インターフェイスがあるので、ピクセルのデータ型は 16UC1、1 クロックごとに最大 4 ピクセル (XF\_NPPC4) が可能です。

## 絶対差分 (absdiff)

`absdiff` 関数は、2 つの入力画像のビット単位の絶対差分を算出し、出力画像を返します。入力および出力画像は XF\_8UC1 タイプである必要があります。

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

説明:

- $I_{out}(x, y)$ : 出力画像の (x,y) 位置での強度。
- $I_{in1}(x, y)$ : 最初の入力画像の (x,y) 位置での強度。
- $I_{in2}(x, y)$ : 2 つ目の入力画像の (x,y) 位置での強度。

### API 構文

```
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 )
```

### パラメーターの説明

次の表に、テンプレートと関数のパラメーターを説明します。

表 30: `absdiff` 関数のパラメーターの説明

パラメーター	説明
SRC_T	入力および出力のピクセルのデータ型。8 ビット、符号なし、1 および 3 チャンネル (XF_8UC1、XF_8UC3) のみサポート。
ROWS	入力および出力画像の最大高さ。
COLS	入力および出力画像の最大幅。8 ピクセル動作の場合は 8 の倍数で指定。
NPC	1 サイクルごとに処理されるピクセル数。1 ピクセルの場合は XF_NPPC1、8 ピクセルの場合は XF_NPPC8。
src1	入力画像
src2	入力画像
dst	出力画像

### リソース使用量

次の表に、Xczu9eg-ffvb1156-1-i-es1 でグレースケール HD (1080x1920) 画像を処理するために、Vivado HLS 2019.1 ツールで生成された異なる設定でのリソース使用量を示します。

表 31: absdiff 関数のリソース使用量のサマリ

動作モード	動作周波数 (MHz)	使用量の見積もり				
		BRAM_18K	DSP_48E	FF	LUT	CLB
1 ピクセル	300	0	0	62	67	17
8 ピクセル	150	0	0	67	234	39

### パフォーマンス見積もり

次の表に、Xczu9eg-ffvb1156-1-i-es1 でグレースケール HD (1080x1920) 画像を処理するために、Vivado HLS 2019.1 ツールで生成された異なる設定でのパフォーマンス見積もりを示します。

表 32: absdiff 関数のパフォーマンス見積もりのサマリ

動作モード	レイテンシ見積もり
	最大レイテンシ (ms)
1 ピクセル動作 (300 MHz)	6.9
8 ピクセル動作 (150 MHz)	1.69

### OpenCV との違い

absdiff 関数では 8 ビット ピクセルがサポートされることを除き、OpenCV と同じです。

## 累積 (accumulate)

accumulate 関数は、画像 (src1) をアキュムレータ画像 (src2) に追加し、累積結果 (dst) を生成します。

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

### API 構文

```
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 )
```

### パラメーターの説明

次の表に、テンプレートと関数のパラメーターを説明します。

表 33: accumulate 関数のパラメーターの説明

パラメーター	説明
SRC_T	入力ピクセルのデータ型。8 ビット、符号なし、1 および 3 チャンネル (XF_8UC1、XF_8UC3) のみサポート。
DST_T	出力ピクセルのデータ型。16 ビット、符号なし、1 および 3 チャンネル (XF_16UC1、XF_16UC3) のみサポート。
ROWS	入力および出力画像の最大高さ。



表 33: accumulate 関数のパラメーターの説明 (続き)

パラメーター	説明
COLS	入力および出力画像の最大幅。8 ピクセル動作の場合は 8 の倍数で指定することを推奨。
NPC	1 サイクルごとに処理されるピクセル数。1 ピクセルの場合は XF_NPPC1、8 ピクセルの場合は XF_NPPC8。
src1	入力画像
src2	入力画像
dst	出力画像

## リソース使用量

次の表に、Xczu9eg-ffvb1156-1-i-es1 でグレースケール HD (1080x1920) 画像を処理するために、Vivado HLS 2019.1 ツールで生成された異なる設定でのリソース使用量を示します。

表 34: accumulate 関数のリソース使用量のサマリ

動作モード	動作周波数 (MHz)	使用量の見積もり				
		BRAM_18K	DSP_48E	FF	LUT	CLB
1 ピクセル	300	0	0	62	55	12
8 ピクセル	150	0	0	389	285	61

次の表に、Xczu9eg-ffvb1156-1-i-es1 で 3 チャンネルの 4K 画像を処理するために、Vivado HLS 2019.1 ツールで生成された異なる設定でのリソース使用量を示します。

表 35: accumulate 関数のリソース使用量のサマリ

動作モード	動作周波数 (MHz)	使用量の見積もり				
		BRAM_18K	DSP_48E	FF	LUT	CLB
1 ピクセル	300	0	1	207	72	32

## パフォーマンス見積もり

次の表に、Xczu9eg-ffvb1156-1-i-es1 でグレースケール HD (1080x1920) 画像を処理するために、Vivado HLS 2019.1 ツールで生成された異なる設定でのパフォーマンス見積もりを示します。

表 36: accumulate 関数のパフォーマンス見積もりのサマリ

動作モード	レイテンシ見積もり
	最大レイテンシ (ms)
1 ピクセル動作 (300 MHz)	6.9
8 ピクセル動作 (150 MHz)	1.7

## OpenCV との違い

OpenCV では、累積画像は 2 番目の入力画像に格納されます。src2 画像は、次に示すように入力および出力の両方として使用されます。

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

xfOpenCV インプリメンテーションでは、累積画像は次に示すように別に格納されます。

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

## 累積二乗 (accumulateSquare)

accumulateSquare 関数は、画像 (src1) をアキュムレータ画像 (src2) に追加し、累積結果 (dst) を生成します。

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

src2 を累積結果とするのではなく、別の引数を使用されます。このインプリメンテーションではストリームが使用されるので、双方向アキュムレータは使用できません。

### API 構文

```
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)
```

### パラメーターの説明

次の表に、テンプレートと関数のパラメーターを説明します。

表 37: accumulateSquare 関数のパラメーターの説明

パラメーター	説明
SRC_T	入力ピクセルのデータ型。8 ビット、符号なし、1 および 3 チャンネル (XF_8UC1、XF_8UC3) のみサポート。
DST_T	出力ピクセルのデータ型。16 ビット、符号なし、1 および 3 チャンネル (XF_16UC1、XF_16UC3) のみサポート。
ROWS	入力および出力画像の最大高さ。
COLS	入力および出力画像の最大幅 (8 ピクセル動作の場合は 8 の倍数で指定)
NPC	1 サイクルごとに処理されるピクセル数。1 ピクセルの場合は XF_NPPC1、8 ピクセルの場合は XF_NPPC8。
src1	入力画像
src2	入力画像
dst	出力画像

### リソース使用量

次の表に、Xczu9eg-ffvb1156-1-i-es1 でグレースケール HD (1080x1920) 画像を処理するために、Vivado HLS 2019.1 ツールで生成された異なる設定でのリソース使用量を示します。

表 38: accumulateSquare 関数のリソース使用量のサマリ

動作モード	動作周波数 (MHz)	使用量の見積もり				
		BRAM_18K	DSP_48E	FF	LUT	CLB
1 ピクセル	300	0	1	71	52	14
8 ピクセル	150	0	8	401	247	48

次の表に、Xczu9eg-ffvb1156-1-i-es1 FPGA で 3 チャンネルの 4K 画像を処理するために、Vivado HLS 2019.1 ツールで生成された異なる設定でのリソース使用量を示します。

表 39: accumulateSquare 関数のリソース使用量のサマリ

動作モード	動作周波数 (MHz)	使用量の見積もり				
		BRAM_18K	DSP_48E	FF	LUT	CLB
1 ピクセル	300	0	3	227	86	37

### パフォーマンス見積もり

次の表に、Xczu9eg-ffvb1156-1-i-es1 でグレースケール HD (1080x1920) 画像を処理するために、Vivado HLS 2019.1 ツールで生成された異なる設定でのパフォーマンス見積もりを示します。

表 40: accumulateSquare 関数のパフォーマンス見積もりのサマリ

動作モード	レイテンシ見積もり
	最大レイテンシ (ms)
1 ピクセル動作 (300 MHz)	6.9
8 ピクセル動作 (150 MHz)	1.6

### OpenCV との違い

OpenCV では、累積二乗画像は 2 番目の入力画像に格納されます。src2 画像は入力および出力の両方として使用されます。

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

xfOpenCV インプリメンテーションでは、累積二乗画像は別に格納されます。

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

## 累積重み (accumulateWeighted)

accumulateWeighted 関数は、入力画像 (src1) とアキュムレータ画像 (src2) の加重和を計算し、結果を dst に生成します。

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

src2 を累積結果とするのではなく、別の引数を使用されます。このインプリメンテーションではストリームが使用されるので、双方向アキュムレータは使用できません。

## API 構文

```
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 )
```

## パラメーターの説明

次の表に、テンプレートと関数のパラメーターを説明します。

表 41: accumulateWeighted 関数のパラメーターの説明

パラメーター	説明
SRC_T	入力ピクセルのデータ型。8 ビット、符号なし、1 および 3 チャンネル (XF_8UC1、XF_8UC3) のみサポート。
DST_T	出力ピクセルのデータ型。16 ビット、符号なし、1 および 3 チャンネル (XF_16UC1、XF_16UC3) のみサポート。
ROWS	入力および出力画像の最大高さ。
COLS	入力および出力画像の最大幅。8 ピクセル動作の場合は 8 の倍数で指定することを推奨。
NPC	1 サイクルごとに処理されるピクセル数。1 ピクセルの場合は XF_NPPC1、8 ピクセルの場合は XF_NPPC8。
src1	入力画像
src2	入力画像
dst	出力画像
alpha	入力画像に適用される重み

## リソース使用量

次の表に、Xczu9eg-ffvb1156-1-i-es1 でグレースケール HD (1080x1920) 画像を処理するために、Vivado HLS 2019.1 ツールで生成された異なる設定でのリソース使用量を示します。

表 42: accumulateWeighted 関数のリソース使用量のサマリ

動作モード	動作周波数 (MHz)	使用量の見積もり				
		BRAM_18K	DSP_48E	FF	LUT	CLB
1 ピクセル	300	0	5	295	255	52
8 ピクセル	150	0	19	556	476	88

次の表に、Xczu9eg-ffvb1156-1-i-es1 FPGA で 3 チャンネルの 4K 画像を処理するために、Vivado HLS 2019.1 ツールで生成された異なる設定でのリソース使用量を示します。

表 43: accumulateWeighted 関数のリソース使用量のサマリ

動作モード	動作周波数 (MHz)	使用量の見積もり				
		BRAM_18K	DSP_48E	FF	LUT	CLB
1 ピクセル	300	0	9	457	387	95

### パフォーマンス見積もり

次の表に、Xczu9eg-ffvb1156-1-i-es1 でグレースケール HD (1080x1920) 画像を処理するために、Vivado HLS 2019.1 ツールで生成された異なる設定でのパフォーマンス見積もりを示します。

表 44: accumulateWeighted 関数のパフォーマンス見積もりのサマリ

動作モード	レイテンシ見積もり
	最大レイテンシ (ms)
1 ピクセル動作 (300 MHz)	6.9
8 ピクセル動作 (150 MHz)	1.7

### OpenCV との違い

OpenCV の結果の画像が 2 つ目の入力画像に格納されます。src2 画像は次に示すように入力および出力の両方として使用されます。

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

xfOpenCV でインプリメンテーションでは、累積重み画像は別に格納されます。

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

## スカラー値との加算 (addS)

addS 関数は、入力画像 src のピクセルと指定のスカラー値 scl を加算し、結果を dst に保存します。

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

(x,y) はピクセルの空間座標です。

### API 構文

```
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)
```

### パラメーターの説明

次の表に、テンプレートと関数のパラメーターを説明します。

表 45: addS 関数のパラメーターの説明

パラメーター	説明
SRC_T	入力ピクセルのデータ型。8 ビット、符号なし、1 チャンネル (XF_8UC1) をサポート。
ROWS	入力および出力画像の最大高さ。
COLS	入力および出力画像の最大幅。N ピクセルの並列処理の場合、幅は N の倍数で指定。
NPC	1 サイクルごとに処理されるピクセル数。1 ピクセルの場合は XF_NPPC1、8 ピクセルの場合は XF_NPPC8。
_src1	1 つ目の入力画像
_scl	入カスカラー値。サイズはチャンネル数。
_dst	出力画像

### リソース使用量

次の表に、Xczu9eg-ffvb1156-1-i-es1 FPGA 用に Vivado HLS 2019.1 ツールを使用して生成された Resource Optimized (8 ピクセル) モードと Normal モードの AddS 関数のリソース使用量を示します。

表 46: addS 関数のリソース使用量のサマリ

名前	リソース使用量	
	1 クロックごとに 1 ピクセル動作	8 クロックごとに 1 ピクセル動作
	300 MHz	150 MHz
BRAM_18K	0	0
DSP48E	0	0
FF	100	101
LUT	52	185
CLB	20	45

### パフォーマンス見積もり

次の表に、Xczu9eg-ffvb1156-1-i-es1 FPGA でグレースケール HD (1080x1920) 画像を処理するために、Vivado HLS 2019.1 ツールを使用して生成された異なる設定でのカーネルのパフォーマンス見積もりを示します。

表 47: addS 関数のパフォーマンス見積もりのサマリ

動作モード	レイテンシ見積もり	
	動作周波数 (MHz)	レイテンシ (ms)
1 ピクセル	300	6.9
8 ピクセル	150	1.7

## 加重和 (addWeighted)

addWeighted 関数は、2 つの入力画像 src1 と src2 の加重和を計算し、結果を dst に保存します。

$$\text{dst}(x,y) = \text{src1}(x,y) * \alpha + \text{src2}(x,y) * \beta + \gamma$$

## API 構文

```
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)
```

## パラメーターの説明

次の表に、テンプレートと関数のパラメーターを説明します。

表 48: addWeighted 関数のパラメーターの説明

パラメーター	説明
SRC_T	入力ピクセルのデータ型。8 ビット、符号なし、1 チャンネル (XF_8UC1) をサポート。
DST_T	出力ピクセルのデータ型。8 ビット、符号なし、1 チャンネル (XF_8UC1) をサポート。
ROWS	入力および出力画像の最大高さ
COLS	入力および出力画像の最大幅。N ピクセルの並列処理の場合、幅は N の倍数で指定。
NPC	1 サイクルごとに処理されるピクセル数。1 ピクセルの場合は XF_NPPC1、8 ピクセルの場合は XF_NPPC8。
_src1	1 つ目の入力画像
Alpha	1 つ目の入力画像に適用される重み
_src2	2 つ目の入力画像
ベータ	2 つ目の入力画像に適用される重み
gamma	各和に追加するスカラー値
_dst	出力画像

## リソース使用量

次の表に、Xczu9eg-ffvb1156-1-i-es1 FPGA で Vivado HLS 2019.1 バージョンツールを使用して生成された、リソース最適化 (8 ピクセル) モードおよび通常モードにおける addWeighted 関数のリソース使用量を示します。

表 49: addWeighted 関数のリソース使用量のサマリ

名前	リソース使用量	
	1 クロックごとに 1 ピクセル動作	8 クロックごとに 1 ピクセル動作
	300 MHz	150 MHz
BRAM_18K	0	0
DSP48E	11	25
FF	903	680
LUT	851	1077
CLB	187	229

## パフォーマンス見積もり

次の表に、Xczu9eg-ffvb1156-1-i-es1 FPGA でグレースケール HD (1080x1920) 画像を処理するために、Vivado HLS 2019.1 ツールを使用して生成された異なる設定でのカーネルのパフォーマンス見積もりを示します。

表 50: addWeighted 関数のパフォーマンス見積もりのサマリ

動作モード	レイテンシ見積もり	
	動作周波数 (MHz)	レイテンシ (ms)
1 ピクセル	300	6.9
8 ピクセル	150	1.7

## バイラテラル フィルター (bilateralFilter)

通常、平滑化フィルターは画像のエッジに影響する画像を平滑化します。平滑化処理中にエッジを保持するには、バイラテラル フィルターを使用できます。バイラテラル フィルターでは、ガウシアン フィルターと同様に、近傍ピクセルが考慮され、各ピクセルには重みが設定されます。これらの重みには2つの要素があります。1つ目の要素は、ガウシアン フィルターで使用される重みと同じです。2つ目の要素では、近傍ピクセルと評価されるピクセルの強度の違いが考慮されます。

画像に適用されるバイラテラル フィルターは、次のとおりです。

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

ここで、

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

および  $G_{\sigma}$  は分散  $\sigma$  のガウシアン フィルターです。

ガウシアン フィルターは、次の式で求められます。  $G_{\sigma} = e^{-\frac{(x^2 + y^2)}{2\sigma^2}}$

### API 構文

```
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 )
```

### パラメーターの説明

次の表に、テンプレートと関数のパラメーターを説明します。

表 51: bilateralFilter 関数のパラメーターの説明

パラメーター	説明
FILTER_SIZE	フィルター サイズ。サポートされるフィルター サイズは 3 (XF_FILTER_3X3)、5 (XF_FILTER_5X5)、および 7 (XF_FILTER_7X7)。
BORDER_TYPE	サポートされる境界タイプは XF_BORDER_CONSTANT。
TYPE	入力および出力のピクセルのデータ型。8 ビット、符号なし、1 チャンネルおよび 3 チャンネル (XF_8UC1、XF_8UC3) のみサポート。



表 51: bilateralFilter 関数のパラメーターの説明 (続き)

パラメーター	説明
ROWS	入力および出力画像の最大高さ。
COLS	入力および出力画像の最大幅 (8 ピクセル動作の場合は 8 の倍数で指定)
NPC	1 サイクルごとに処理されるピクセル数。XF_NPPC1 および XF_NPPC8 をサポート。
src	入力画像
dst	出力画像
sigma_space	空間領域のフィルターの標準偏差
sigma_color	色空間で使用するフィルターの標準偏差

### リソース使用量

次の表に、Xczu9eg-ffvb1156-1-i-es1 FPGA でグレースケール HD (1080x1920) 画像を処理するために、Vivado HLS 2019.1 ツールを使用して生成された異なる設定でのカーネルのリソース使用量を示します。

表 52: bilateralFilter 関数のリソース使用量のサマリ

動作モード	フィルター サイズ	動作周波数 (MHz)	使用量の見積もり			
			BRAM_18K	DSP_48E	FF	LUT
1 ピクセル	3x3	300	6	22	4934	4293
	5x5	300	12	30	5481	4943
	7x7	300	37	48	7084	6195

次の表に、Xczu9eg-ffvb1156-1-i-es1 FPGA で 3 チャンネルの 4K 画像を処理するために、Vivado HLS 2019.1 ツールを使用して生成された異なる設定でのカーネルのリソース使用量を示します。

表 53: bilateralFilter 関数のリソース使用量のサマリ

動作モード	フィルター サイズ	動作周波数 (MHz)	使用量の見積もり			
			BRAM_18K	DSP_48E	FF	LUT
1 ピクセル	3x3	300	12	32	8342	7442
	5x5	300	27	57	10663	8857
	7x7	300	49	107	12870	12181

### パフォーマンス見積もり

次の表に、Xczu9eg-ffvb1156-1-i-es1 FPGA でグレースケール HD (1080x1920) 画像を処理するために、Vivado HLS 2019.1 ツールを使用して生成された異なる設定でのカーネルのパフォーマンス見積もりを示します。

表 54: bilateralFilter 関数のパフォーマンス見積もりのサマリ

動作モード	フィルター サイズ	レイテンシ見積もり
		300 MHz
		最大 (ms)
1 ピクセル	3x3	7.18
	5x5	7.20
	7x7	7.22

## OpenCV との違い

OpenCV とは異なり、xfOpenCV ではフィルター サイズ 3、5、および 7 のみがサポートされます。

## ビット深度変換 (convertTo)

convertTo 関数は、入力画像のビット深度を出力画像に必要なビット深度に変換します。

### API 構文

```
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)
```

### パラメーターの説明

次の表に、テンプレートと関数のパラメーターを説明します。

表 55: convertTo 関数のパラメーターの説明

パラメーター	説明
SRC_T	入力ピクセルのデータ型。8 ビット、符号なし、1 チャンネル (XF_8UC1)、16 ビット、符号なし、1 チャンネル (XF_16UC1)、16 ビット、符号付き、1 チャンネル (XF_16SC1)、32 ビット、符号なし、1 チャンネル (XF_32UC1)、32 ビット、符号付き、1 チャンネル (XF_32SC1) をサポート。
DST_T	出力ピクセルのデータ型。8 ビット、符号なし、1 チャンネル (XF_8UC1)、16 ビット、符号なし、1 チャンネル (XF_16UC1)、16 ビット、符号付き、1 チャンネル (XF_16SC1)、32 ビット、符号なし、1 チャンネル (XF_32UC1)、32 ビット、符号付き、1 チャンネル (XF_32SC1) をサポート。
ROWS	入力および出力画像の高さ。
COLS	入力および出力画像の幅。
NPC	1 サイクルごとに処理されるピクセル数。1 ピクセルの場合は XF_NPPC1、8 ピクセルの場合は XF_NPPC8。XF_NPPC8 は、32 ビット入力および出力のピクセルのデータ型ではサポートされません。
_src_mat	入力画像
_dst_mat	出力画像
_convert_type	必要な変換タイプを指定。有効な値は、xf_params.h ファイルの XF_convert_bit_depth_e 列挙型を参照。

表 55: convertTo 関数のパラメーターの説明 (続き)

パラメーター	説明
_shift	オブションのスケール係数

### 可能な変換

次の表に、サポートされる変換を示します。可能な入力画像ビット深度を行に、可能な出力画像ビット深度を列に示します (U = 符号なし、S = 符号付き)。

表 56: convertTo 関数でサポートされる変換

入力/出力	U8	U16	S16	U32	S32
U8	なし	可	可	なし	可
U16	可	なし	なし	なし	可
S16	可	なし	なし	なし	可
U32	なし	なし	なし	なし	なし
S32	可	可	可	なし	なし

### リソース使用量

次の表に、サイリンクス Xczu9eg-ffvb1156-1-i-es1 FPGA でグレースケール HD (1080x1920) 画像を処理するために、Vivado HLS 2019.1 ツールを使用して生成された convertTo 関数のリソース使用量を示します。

表 57: XF\_CONVERT\_8U\_TO\_16S 変換のための convertTo 関数のリソース使用量のサマリ

動作モード	動作周波数 (MHz)	使用量の見積もり				
		BRAM_18K	DSP_48E	FF	LUT	CLB
1 ピクセル	300	0	8	581	523	119
8 ピクセル	150	0	8	963	1446	290

表 58: XF\_CONVERT\_16U\_TO\_8U 変換のための convertTo 関数のリソース使用量のサマリ

動作モード	動作周波数 (MHz)	使用量の見積もり				
		BRAM_18K	DSP_48E	FF	LUT	CLB
1 ピクセル	300	0	8	591	541	124
8 ピクセル	150	0	8	915	1500	308

### パフォーマンス見積もり

次の表に、Xczu9eg-ffvb1156-1-i-es1 でグレースケール HD (1080x1920) 画像を処理するために、Vivado HLS 2019.1 ツールで生成された異なる設定でのパフォーマンス見積もりを示します。

表 59: convertTo 関数のパフォーマンス見積りのサマリ

動作モード	レイテンシ見積もり
	最大レイテンシ
1 ピクセル動作 (300 MHz)	6.91 ms
8 ピクセル動作 (150 MHz)	1.69 ms

## ビット単位 AND (bitwise\_and)

bitwise\_and 関数は、2つの画像の各ピクセルに対してビット単位の AND 演算を実行し、出力画像を返します。

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

説明:

- ・  $I_{out}(x, y)$  : 出力画像の (x,y) 位置での強度
- ・  $I_{in1}(x, y)$  : 1 目の入力画像の (x,y) 位置での強度
- ・  $I_{in2}(x, y)$  : 2 目の入力画像の (x,y) 位置での強度

### API 構文

```
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 )
```

### パラメーターの説明

次の表に、テンプレートと関数のパラメーターを説明します。

表 60: bitwise\_and 関数のパラメーターの説明

パラメーター	説明
SRC_T	入力および出力のピクセルのデータ型。1 チャンネルおよび 3 チャンネル (XF_8UC1、XF_8UC3) をサポート。
ROWS	入力および出力画像の最大高さ。
COLS	入力および出力画像の最大幅 (8 ピクセル モードの場合は 8 の倍数で指定)。
NPC	1 サイクルごとに処理されるピクセル数。1 ピクセル動作の場合は XF_NPPC1、8 ピクセル動作の場合は XF_NPPC8。
src1	入力画像
src2	入力画像
dst	出力画像

### リソース使用量

次の表に、Xczu9eg-ffvb1156-1-i-es1 でグレースケール HD (1080x1920) 画像を処理するために、Vivado HLS 2019.1 ツールで生成された異なる設定でのリソース使用量を示します。

表 61: bitwise\_and 関数のリソース使用量のサマリ

動作モード	動作周波数 (MHz)	使用量の見積もり				
		BRAM_18K	DSP_48E	FF	LUT	CLB
1 ピクセル	300	0	0	62	44	10

表 61: bitwise\_and 関数のリソース使用量のサマリ (続き)

動作モード	動作周波数 (MHz)	使用量の見積もり				
		BRAM_18K	DSP_48E	FF	LUT	CLB
8 ピクセル	150	0	0	59	72	13

次の表に、Xczu9eg-ffvb1156-1-i-es1 FPGA で 3 チャンネルの 4K 画像を処理するために、Vivado HLS 2019.1 ツールで生成された異なる設定でのリソース使用量を示します。

表 62: bitwise\_and 関数のリソース使用量のサマリ

動作モード	動作周波数 (MHz)	使用量の見積もり				
		BRAM_18K	DSP_48E	FF	LUT	CLB
1 ピクセル	300	0	1	155	61	22

### パフォーマンス見積もり

次の表に、Xczu9eg-ffvb1156-1-i-es1 でグレースケール HD (1080x1920) 画像を処理するために、Vivado HLS 2019.1 ツールで生成された異なる設定でのパフォーマンス見積もりを示します。

表 63: bitwise\_and 関数のパフォーマンス見積もりのサマリ

動作モード	レイテンシ見積もり
	最大レイテンシ (ms)
1 ピクセル動作 (300 MHz)	6.9
8 ピクセル動作 (150 MHz)	1.7

## ビット単位 NOT (bitwise\_not)

bitwise\_not 関数は、入力画像のピクセルに対してビット単位の NOT 演算を実行し、出力画像を返します。

$$I_{out}(x, y) = \sim I_{in}(x, y)$$

説明:

- ・  $I_{out}(x, y)$  : 出力画像の (x, y) 位置での強度
- ・  $I_{in}(x, y)$  : 入力画像の (x, y) 位置での強度

### API 構文

```
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 )
```

## パラメーターの説明

次の表に、テンプレートと関数のパラメーターを説明します。

表 64: bitwise\_not 関数のパラメーターの説明

パラメーター	説明
SRC_T	入力および出力のピクセルのデータ型。1 チャンネルおよび 3 チャンネル (XF_8UC1、XF_8UC3) をサポート。
ROWS	入力および出力画像の最大高さ。
COLS	入力および出力画像の最大幅。8 ピクセル モードの場合は 8 の倍数で指定。
NPC	1 サイクルごとに処理されるピクセル数。1 ピクセル動作の場合は XF_NPPC1、8 ピクセル動作の場合は XF_NPPC8。
src	入力画像
dst	出力画像

## リソース使用量

次の表に、Xczu9eg-ffvb1156-1-i-es1 でグレースケール HD (1080x1920) 画像を処理するために、Vivado HLS 2019.1 ツールで生成された異なる設定でのリソース使用量を示します。

表 65: bitwise\_not 関数のリソース使用量のサマリ

動作モード	動作周波数 (MHz)	使用量の見積もり				
		BRAM_18K	DSP_48E	FF	LUT	CLB
1 ピクセル	300	0	0	97	78	20
8 ピクセル	150	0	0	88	97	21

次の表に、Xczu9eg-ffvb1156-1-i-es1 FPGA で 3 チャンネルの 4K 画像を処理するために、Vivado HLS 2019.1 ツールで生成された異なる設定でのリソース使用量を示します。

表 66: bitwise\_not 関数のリソース使用量のサマリ

動作モード	動作周波数 (MHz)	使用量の見積もり				
		BRAM_18K	DSP_48E	FF	LUT	CLB
1 ピクセル	300	0	1	155	61	22

## パフォーマンス見積もり

次の表に、Xczu9eg-ffvb1156-1-i-es1 でグレースケール HD (1080x1920) 画像を処理するために、Vivado HLS 2019.1 ツールで生成された異なる設定でのパフォーマンス見積もりを示します。

表 67: bitwise\_not 関数のパフォーマンス見積もりのサマリ

動作モード	レイテンシ見積もり
	最大レイテンシ (ms)
1 ピクセル動作 (300 MHz)	6.9
8 ピクセル動作 (150 MHz)	1.7

## ビット単位 OR (bitwise\_or)

bitwise\_or 関数は、2つの入力画像のビット単位の OR 演算を実行し、出力画像を返します。

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

説明:

- ・  $I_{out}(x, y)$  : 出力画像の (x,y) 位置での強度
- ・  $I_{in1}(x, y)$  : 1つ目の入力画像の (x,y) 位置での強度
- ・  $I_{in2}(x, y)$  : 2つ目の入力画像の (x,y) 位置での強度

### API 構文

```
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 )
```

### パラメーターの説明

次の表に、テンプレートと関数のパラメーターを説明します。

表 68: bitwise\_or 関数のパラメーターの説明

パラメーター	説明
SRC_T	入力および出力のピクセルのデータ型。1 チャンネルおよび 3 チャンネル (XF_8UC1、XF_8UC3) をサポート。
ROWS	入力および出力画像の最大高さ。
COLS	入力および出力画像の最大幅。8 ピクセル モードの場合は 8 の倍数で指定。
NPC	1 サイクルごとに処理されるピクセル数。1 ピクセルの場合は XF_NPPC1、8 ピクセルの場合は XF_NPPC8。
src1	入力画像
src2	入力画像
dst	出力画像

### リソース使用量

次の表に、Xczu9eg-ffvb1156-1-i-es1 でグレースケール HD (1080x1920) 画像を処理するために、Vivado HLS 2019.1 ツールで生成された異なる設定でのリソース使用量を示します。



表 69: bitwise\_or 関数のリソース使用量のサマリ

動作モード	動作周波数 (MHz)	使用量の見積もり				
		BRAM_18K	DSP_48E	FF	LUT	CLB
1 ピクセル	300	0	0	62	44	10
8 ピクセル	150	0	0	59	72	13

次の表に、Xczu9eg-ffvb1156-1-i-es1 FPGA で 3 チャンネルの 4K 画像を処理するために、Vivado HLS 2019.1 ツールで生成された異なる設定でのリソース使用量を示します。

表 70: bitwise\_or 関数のリソース使用量のサマリ

動作モード	動作周波数 (MHz)	使用量の見積もり				
		BRAM_18K	DSP_48E	FF	LUT	CLB
1 ピクセル	300	0	1	155	61	22

### パフォーマンス見積もり

次の表に、Xczu9eg-ffvb1156-1-i-es1 でグレースケール HD (1080x1920) 画像を処理するために、Vivado HLS 2019.1 ツールで生成された異なる設定でのパフォーマンス見積もりを示します。

表 71: bitwise\_or 関数のパフォーマンス見積もりのサマリ

動作モード	レイテンシ見積もり
	最大レイテンシ (ms)
1 ピクセル動作 (300 MHz)	6.9
8 ピクセル動作 (150 MHz)	1.7

## ビット単位 XOR (bitwise\_xor)

bitwise\_xor 関数は、2 つの入力画像のビット単位の XOR 演算を実行し、出力画像を返します。

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

説明:

- $I_{out}(x, y)$  : 出力画像の (x, y) 位置での強度
- $I_{in1}(x, y)$  : 1 つ目の入力画像の (x, y) 位置での強度
- $I_{in2}(x, y)$  : 2 つ目の入力画像の (x, y) 位置での強度

## API 構文

```
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 )
```

## パラメーターの説明

次の表に、テンプレートと関数のパラメーターを説明します。

表 72: bitwise\_xor 関数のパラメーターの説明

パラメーター	説明
SRC_T	入力および出力のピクセルのデータ型。1 チャンネルおよび 3 チャンネル (XF_8UC1、XF_8UC3) をサポート。
ROWS	入力および出力画像の最大高さ。
COLS	入力および出力画像の最大幅。8 ピクセル モードの場合は 8 の倍数で指定。
NPC	1 サイクルごとに処理されるピクセル数。1 ピクセルの場合は XF_NPPC1、8 ピクセルの場合は XF_NPPC8。
src1	入力画像
src2	入力画像
dst	出力画像

## リソース使用量

次の表に、Xczu9eg-ffvb1156-1-i-es1 FPGA でグレースケール HD (1080x1920) 画像を処理するために、Vivado HLS 2019.1 ツールを使用して生成された異なる設定でのリソース使用量を示します。

表 73: bitwise\_xor 関数のリソース使用量のサマリ

動作モード	動作周波数 (MHz)	使用量の見積もり				
		BRAM_18K	DSP_48E	FF	LUT	CLB
1 ピクセル	300	0	0	62	44	10
8 ピクセル	150	0	0	59	72	13

次の表に、Xczu9eg-ffvb1156-1-i-es1 FPGA で 4K チャンネル画像を処理するために、Vivado HLS 2019.1 ツールで生成された異なる設定でのリソース使用量を示します。

表 74: bitwise\_xor 関数のリソース使用量のサマリ

動作モード	動作周波数 (MHz)	使用量の見積もり				
		BRAM_18K	DSP_48E	FF	LUT	CLB
1 ピクセル	300	0	1	155	61	22

## パフォーマンス見積もり

次の表に、Xczu9eg-ffvb1156-1-i-es1 でグレースケール HD (1080x1920) 画像処理するために、Vivado HLS 2019.1 ツールを使用して生成された異なる設定でのパフォーマンス見積もりを示します。

表 75: bitwise\_xor 関数のパフォーマンス見積もりのサマリ

動作モード	レイテンシ見積もり
	最大レイテンシ (ms)
1 ピクセル動作 (300 MHz)	6.9
8 ピクセル動作 (150 MHz)	1.7

## ボックス フィルター (boxFilter)

boxFilter 関数は、入力画像に対してボックス フィルター処理を実行します。ボックス フィルターローパス フィルターとして機能し、画像にぼかしを適用します。boxFilter 関数 (ボックスぼかし) は空間領域線形フィルターで、処理後の画像の各ピクセルは近傍ピクセルの平均値となります。

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

### API 構文

```
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)
```

### パラメーターの説明

次の表に、テンプレートと関数のパラメーターを説明します。

表 76: boxFilter 関数のパラメーターの説明

パラメーター	説明
FILTER_SIZE	フィルター サイズ。サポートされるフィルター サイズは 3 (XF_FILTER_3X3)、5 (XF_FILTER_5X5)、および 7 (XF_FILTER_7X7)。
BORDER_TYPE	サポートされる境界タイプは XF_BORDER_CONSTANT。
SRC_T	入力および出力のピクセルのデータ型。8 ビット、符号なし、16 ビット符号なし、および 16 ビット符号付き 1 チャンネル (XF_8UC1) をサポート。
ROWS	入力および出力画像の最大高さ。
COLS	入力および出力画像の最大幅 (8 ピクセル動作の場合は 8 の倍数で指定)
NPC	1 サイクルごとに処理されるピクセル数。1 ピクセルの場合は XF_NPPC1、8 ピクセルの場合は XF_NPPC8。
USE_URAM	ストレージ構造を UltraRAM にマップ
_src_mat	入力画像
_dst_mat	出力画像

## リソース使用量

次の表に、Xczu9eg-ffvb1156-1-i-es1 FPGA でグレースケール HD (1080x1920) 画像を処理するために、Vivado HLS 2019.1 ツールを使用して生成された異なる設定でのカーネルのリソース使用量を示します。

表 77: boxFilter 関数のリソース使用量のサマリ

動作モード	フィルター サイズ	動作周波数 (MHz)	使用量の見積もり				
			BRAM_18K	DSP_48E	FF	LUT	CLB
1 ピクセル	3x3	300	3	1	545	519	104
	5x5	300	5	1	876	870	189
	7x7	300	7	1	1539	1506	300
8 ピクセル	3x3	150	6	8	1002	1368	264
	5x5	150	10	8	1576	3183	611
	7x7	150	14	8	2414	5018	942

次の表に、xczu7ev-ffvc1156-2-e FPGA で UltraRAM をイネーブルにしてグレースケール 4K (3840x2160) 画像を処理するために、SDx™ 2019.1 ツールを使用して生成された異なる設定でのカーネルのリソース使用量を示します。

表 78: UltraRAM をイネーブルにした場合の boxfilter 関数のリソース使用量のサマリ

動作モード	フィルター サイズ	動作周波数 (MHz)	使用量の見積もり				
			BRAM_18K	URAM	DSP_48E	FF	LUT
1 ピクセル	3x3	300	0	1	1	821	521
	5x5	300	0	1	1	1204	855
	7x7	300	0	1	1	2083	1431
8 ピクセル	3x3	150	0	3	8	1263	1480
	5x5	150	0	5	8	1771	3154
	7x7	150	0	7	8	2700	5411

## パフォーマンス見積もり

次の表に、Xczu9eg-ffvb1156-1-i-es1 でグレースケール HD (1080x1920) 画像を処理するために、Vivado HLS 2019.1 ツールで生成された異なる設定でのカーネルのパフォーマンス見積もりを示します。

表 79: boxFilter 関数のパフォーマンス見積もりのサマリ

動作モード	動作周波数 (MHz)	フィルター サイズ	レイテンシ見積もり
			最大 (ms)
1 ピクセル	300	3x3	7.2
	300	5x5	7.21
	300	7x7	7.22

表 79: boxFilter 関数のパフォーマンス見積もりのサマリ (続き)

動作モード	動作周波数 (MHz)	フィルター サイズ	レイテンシ見積もり
			最大 (ms)
8 ピクセル	150	3x3	1.7
	150	5x5	1.7
	150	7x7	1.7

## 境界ボックス (BoundingBox)

boundingbox 関数は、入力画像の関心領域 (ROI) をハイライトします。

$$P(X,Y) \leq P(x_i, y_i) \leq P(X,Y')$$

$$P(X',Y) \leq P(x_i, y_i) \leq P(X',Y')$$

説明:

- $P(x_i, y_i)$ : 現在のピクセル位置
- $P(X,Y)$ : ROI の左上角
- $P(X,Y')$ : ROI の右上角
- $P(X',Y)$ : ROI の左下角
- $P(X',Y')$ : ROI の右下角

### API 構文

```
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)
```

### パラメーターの説明

次の表に、テンプレートと関数のパラメーターを説明します。

表 80: boundingbox 関数のパラメーターの説明

パラメーター	説明
SRC_T	入力ピクセルのデータ型。8ビット、符号なし、1チャンネルおよび3チャンネル (XF_8UC1、XF_8UC3) のみサポート。
ROWS	入力および出力画像の最大高さ。
COLS	入力および出力画像の最大幅。NPC の倍数で指定。
MAX_BOXES	ボックスの最大数 (5 に固定)
NPC	1 サイクルごとに処理されるピクセル数。可能なオプションは XF_NPPC1 のみ。
_src_mat	入力画像
roi	矩形の左上角、高さ、および幅で構成される xf::Rect オブジェクト。
color	各ボックス (ROI) の色情報を含む xf::Scalar オブジェクト。

表 80: boundingbox 関数のパラメーターの説明 (続き)

パラメーター	説明
num_box	検出されたボックスの数。MAX_BOXES 以下である必要があります。

### リソース使用量

次の表に、ザイリンクス Xczu9eg-ffvb1156-1-i-es1 FPGA 用に Vivado HLS 2019.1 ツールを使用して生成された異なる設定でのリソース使用量を示します。

表 81: boundingbox 関数のリソース使用量のサマリ

動作モード	動作周波数 (MHz)	使用量の見積もり				
		BRAM_18K	DSP_48E	FF	LUT	CLB
1 ピクセル	300	5	4	2521	1649	409

### パフォーマンス見積もり

次の表に、ザイリンクス xczu9eg-ffvb1156-2-i-es2 FPGA でグレースケール 4K (2160x3840) 画像で 3 つの境界 (480x640、100x200、300x300) をハイライトするために、Vivado HLS 2019.1 ツールを使用して生成された 1 ピクセル設定でのカーネルのパフォーマンス見積もりを示します。

表 82: boundingbox 関数のパフォーマンス見積もりのサマリ

動作モード	レイテンシ見積もり
	最大レイテンシ (ms)
1 ピクセル動作 (300 MHz)	0.15

### xfOpenCV 基準

xf::boundingbox は、次の xfOpenCV 関数に準拠しています。

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

## Canny 法によるエッジ検出 (Canny)

Canny 法によるエッジ検出では、画像またはビデオ フレームのエッジを検出します。エッジ検出で最もよく使用されるアルゴリズムの 1 つです。Canny アルゴリズムでは、次の 3 つの主な条件を満たすことが目的とされます。

1. 低エラー レート: 既存のエッジのみを検出。
2. 局所化: 検出されたエッジピクセルと実際のエッジピクセルとの距離を最小化。
3. 最低限の応答: 各エッジに対して 1 つの検出器のみが応答。

このアルゴリズムでは、まずガウシアン マスクが適用されて画像のノイズが削減されます。ここで使用されるガウシアン マスクは、3x3 サイズの平均マスクです。その後、Sobel 勾配関数を使用して x および y 方向の勾配が計算されます。勾配は、ピクセルの大きさと位相を計算するために使用されます。位相は量子化され、それに応じてピクセルがビンに分類されます。ピクセルに NMS (Non-Maximal Suppression) が適用され、弱いエッジは除去されます。

残りのエッジにエッジトレースが適用され、画像のエッジが描画されます。このアルゴリズムでは、Canny から NMS までは 1 つのカーネルに含まれ、エッジリンクング モジュールは別のカーネルに含まれます。NMS の後、出力に各ピクセルが 2 ビットで表されます。

- 00: 背景
- 01: 弱いエッジ
- 11: 強いエッジ

出力は、1 サイクルごとに 1 ピクセル動作では 8 ビット (4 つの 2 ビット ピクセル) としてパックされ、1 サイクルごとに 8 ピクセル動作では 16 ビット (8 つの 2 ビット ピクセル) としてパックされます。エッジリンクング モジュールでは、入力 は 64 ビットであり、32 個の 2 ビット ピクセルが 64 ビットにパックされます。ピクセルにエッジトレースが適用され、画像のエッジが返されます。

## API 構文

Canny の API 構文は次のとおりです。

```
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)
```

EdgeTracing の API 構文は次のとおりです。

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

## パラメーターの説明

次の表に、xf::Canny のテンプレートと関数のパラメーターを説明します。

表 83: xf::Canny 関数のパラメーターの説明

パラメーター	説明
FILTER_TYPE	フィルター ウィンドウの大きさ。有効な値は 3 および 5。
NORM_TYPE	使用するノルムのタイプ。有効なノルム タイプは L1NORM および L2NORM。
SRC_T	入力ピクセルのデータ型。8 ビット、符号なし、1 チャンネル (XF_8UC1) のみサポート。
DST_T	出力ピクセルのデータ型。XF_2UC1 のみサポート。NPC=XF_NPPC1 の場合の出力は 8 ビットで、4 つの 2 ビット ピクセル値が 8 ビットにパックされます。NPC=XF_NPPC8 の場合の出力は 16 ビットで、8 つの 2 ビット ピクセル値が 16 ビットにパックされます。
ROWS	入力および出力画像の最大高さ
COLS	入力および出力画像の最大幅。8 ピクセル モードの場合は 8 の倍数で指定。
NPC	1 サイクルごとに処理されるピクセル数。1 ピクセルの場合は XF_NPPC1、8 ピクセルの場合は XF_NPPC8。XF_NPPC の場合、出力画像ピクセルはパックされ、精度は XF_NPPC4 です。XF_NPPC8 の場合は出力ピクセルの精度は XF_NPPC4 です。
USE_URAM	一部のストレージ構造を UltraRAM にマップ
_src_mat	入力画像

表 83: xf::Canny 関数のパラメーターの説明 (続き)

パラメーター	説明
_dst_mat	出力画像
_lowthreshold	バイナリしきい値処理の下限しきい値。
_highthreshold	バイナリしきい値処理の上限しきい値。

次の表に、EdgeTracing のテンプレートと関数のパラメーターを説明します。

表 84: EdgeTracing 関数のパラメーターの説明

パラメーター	説明
SRC_T	入力ピクセルのデータ型
DST_T	出力ピクセルのデータ型
ROWS	入力および出力画像の最大高さ
COLS	入力および出力画像の最大幅 (32 の倍数で指定)
NPC_SRC	1 サイクルごとに処理されるピクセル数。XF_NPPC32 に固定。
NPC_DST	デスティネーションに書き込まれたピクセル数。XF_NPPC8 に固定。
USE_URAM	ストレージ構造を UltraRAM にマップ。
_src	入力画像
_dst	出力画像

## リソース使用量

次の表に、Xczu9eg-ffvb1156-1-i-es1 FPGA でグレースケール HD (1080x1920) 画像をフィルター サイズ 3 で処理するために、Vivado HLS 2019.1 ツールを使用して生成された異なる設定での xf::Canny および EdgeTracing のリソース使用量を示します。

表 85: xf::Canny および EdgeTracing 関数のリソース使用量のサマリ

名前	リソース使用量					
	1 ピクセル	1 ピクセル	8 ピクセル	8 ピクセル	エッジ リンクング	エッジ リンクング
	L1NORM、FS:3 300 MHz	L2NORM、FS:3 300 MHz	L1NORM、FS:3 150 MHz	L2NORM、FS:3 150 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

次の表に、xczu7ev-ffvc1156-2-e FPGA でグレースケール 4K 画像をフィルター サイズ 3 で処理するために、SDx 2019.1 ツールを使用して生成された異なる設定での xf::Canny および EdgeTracing のリソース使用量を示します。



表 86: UltraRAM をイネーブルにした場合の xf::Canny および EdgeTracing 関数のリソース使用量のサマリ

名前	リソース使用量					
	1 ピクセル	1 ピクセル	8 ピクセル	8 ピクセル	エッジ リンクング	エッジ リンクング
	L1NORM、FS:3	L2NORM、FS:3	L1NORM、FS:3	L2NORM、FS:3		
	300 MHz	300 MHz	150 MHz	150 MHz	300 MHz	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

### パフォーマンス見積もり

次の表に、Xczu9eg-ffvb1156-1-i-es1 でグレースケール HD (1080x1920) 画像を L1NORM、フィルタ サイズ 3、エッジ リンキング モジュールを使用して処理するために、Vivado HLS 2019.1 ツールを使用して生成された異なる設定でのカーネルのパフォーマンス見積もりを示します。

表 87: xf::Canny および EdgeTracing 関数のパフォーマンス見積もりのサマリ

動作モード	レイテンシ見積もり	
	動作周波数 (MHz)	レイテンシ (ms)
1 ピクセル	300	10.8
8 ピクセル	150	8.5

### OpenCV との違い

OpenCV Canny 関数では、処理前の手順としてガウシアンぼかしは適用されません。

## チャネル結合 (merge)

merge 関数では、シングルチャネル画像がマルチチャネル画像に統合されます。統合されるチャネル数は 4 です。

### API 構文

```
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)
```

### パラメーターの説明

次の表に、テンプレートと関数のパラメーターを説明します。

表 88: merge 関数のパラメーターの説明

パラメーター	説明
SRC_T	入力ピクセルのデータ型。8 ビット、符号なし、1 チャンネル (XF_8UC1) のみサポート。
DST_T	出力ピクセルのデータ型。8 ビット、符号なし、4 チャンネル (XF_8UC4) のみサポート。
ROWS	入力および出力画像の最大高さ。
COLS	入力および出力画像の最大幅。8 ピクセル モードの場合は 8 の倍数で指定。
NPC	1 サイクルごとに処理されるピクセル数。1 ピクセル動作の場合、可能なオプションは XF_NPPC1。
_src1	入力シングルチャンネル画像
_src2	入力シングルチャンネル画像
_src3	入力シングルチャンネル画像
_src4	入力シングルチャンネル画像
_dst	出力マルチチャンネル画像

## リソース使用量

次の表に、Xczu9eg-ffvb1156-1-i-es1 FPGA で 4 つのシングル チャンネル HD (1080x1920) 画像を処理するために、Vivado HLS 2019.1 ツールを使用して生成された merge 関数のリソース使用量を示します。

表 89: merge 関数のリソース使用量のサマリ

動作モード	動作周波数 (MHz)	使用量の見積もり				
		BRAM_18K	DSP_48E	FF	LUT	CLB
1 ピクセル	300	0	8	494	386	85

## パフォーマンス見積もり

次の表に、Xczu9eg-ffvb1156-1-i-es1 で 4 つのシングル チャンネル HD (1080x1920) 画像を処理するために、Vivado HLS 2019.1 ツールで生成された異なる設定でのパフォーマンス見積もりを示します。

表 90: merge 関数のパフォーマンス見積もりのサマリ

動作モード	レイテンシ見積もり
	最大レイテンシ
1 ピクセル動作 (300 MHz)	6.92 ms

## チャンネル抽出 (extractChannel)

extractChannel 関数は、マルチチャンネル配列 (32 ビットのピクセルがインターリーブされたデータ) を複数シングルチャンネル配列に分割し、シングル チャンネルを返します。抽出されるチャンネルは、チャンネル引数を使用して指定します。

チャンネル引数の値は、xf\_channel\_extract\_e 列挙データ型で定義されたマクロにより指定します。次の表に、xf\_channel\_extract\_e 列挙データ型に可能な値を示します。

表 91: xf\_channel\_extract\_e 列挙データ型の値

チャンネル	列挙型
不明	XF_EXTRACT_CH_0
不明	XF_EXTRACT_CH_1
不明	XF_EXTRACT_CH_2
不明	XF_EXTRACT_CH_3
RED	XF_EXTRACT_CH_R
GREEN	XF_EXTRACT_CH_G
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 構文

```
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)
```

## パラメーターの説明

次の表に、テンプレートと関数のパラメーターを説明します。

表 92: extractChannel 関数のパラメーターの説明

パラメーター	説明
SRC_T	入力ピクセルのデータ型。8ビット、符号なし、4チャンネル (XF_8UC4) のみサポート。
DST_T	出力ピクセルのデータ型。8ビット、符号なし、1チャンネル (XF_8UC1) のみサポート。
ROWS	入力および出力画像の最大高さ
COLS	入力および出力画像の最大幅。8ピクセル モードの場合は8の倍数で指定。
NPC	1 サイクルごとに処理されるピクセル数。1ピクセル動作の場合、可能なオプションは XF_NPPC1。
_src_mat	入力マルチチャンネル画像
_dst_mat	出力シングルチャンネル画像
_channel	抽出するチャンネル (有効な値は、xf_params.h ファイルの xf_channel_extract_e 列挙型を参照)

## リソース使用量

次の表に、Xczu9eg-ffvb1156-1-i-es1 FPGA で 4 チャンネル HD (1080x1920) 画像を処理するために、Vivado HLS 2019.1 ツールを使用して生成された extractChannel 関数のリソース使用量を示します。

表 93: extractChannel 関数のリソース使用量のサマリ

動作モード	動作周波数 (MHz)	使用量の見積もり				
		BRAM_18K	DSP_48E	FF	LUT	CLB
1ピクセル	300	0	8	508	354	96

## パフォーマンス見積もり

次の表に、Xczu9eg-ffvb1156-1-i-es1 で 4 チャンネル HD (1080x1920) 画像を処理するために、Vivado HLS 2019.1 ツールで生成された異なる設定でのパフォーマンス見積もりを示します。

表 94: `extractChannel` 関数のパフォーマンス見積もりのサマリ

動作モード	レイテンシ見積もり
	最大レイテンシ (ms)
1 ピクセル動作 (300 MHz)	6.92

## 色変換

色変換関数は、ある画像フォーマットを別の画像フォーマットに変換します。次の表に、変換可能な画像フォーマットの組み合わせを示します。行が入力フォーマットで、列が出力フォーマットです。次のセクションで、サポートされる変換について説明します。

入力/出力 フォーマット	RGBA	NV12	NV21	IYUV	UYVY	YUYV	YUV4	RGB	BGR
RGBA	なし	<a href="#">RGBA to NV12 (rgba2nv12) を参照</a>	<a href="#">RGBA to NV21 (rgba2nv21) を参照</a>	<a href="#">RGBA/RGB to IYUV (rgba2iyuv、rgb2iyuv) を参照</a>			<a href="#">RGBA/RGB to YUV4 (rgba2yuv4、rgb2yuv4) を参照</a>		
NV12	<a href="#">NV12 to RGBA (nv122rgba) を参照</a>	なし	<a href="#">NV12 to NV21、NV21 to NV12 (nv122nv21、nv212nv12) を参照</a>	<a href="#">NV12 to IYUV (nv122iyuv) を参照</a>	<a href="#">NV12/ NV21 to UYVY/ YUYV (nv122uyvy、nv122yuyv、nv212uyvy、nv212yuyv) を参照</a>	<a href="#">NV12/ NV21 to UYVY/ YUYV (nv122uyvy、nv122yuyv、nv212uyvy、nv212yuyv) を参照</a>	<a href="#">NV12 to YUV4 (nv122yuv4) を参照</a>	<a href="#">NV12/ NV21 to RGB/BGR (nv122rgb、nv122bgr、nv212rgb、nv212bgr) を参照</a>	<a href="#">NV12/ NV21 to RGB/BGR (nv122rgb、nv122bgr、nv212rgb、nv212bgr) を参照</a>
NV21	<a href="#">NV21 to RGBA (nv212rgba) を参照</a>	<a href="#">NV12 to NV21、NV21 to NV12 (nv122nv21、nv212nv12) を参照</a>	なし	<a href="#">NV21 to IYUV (nv212iyuv) を参照</a>	<a href="#">NV12/ NV21 to UYVY/ YUYV (nv122uyvy、nv122yuyv、nv212uyvy、nv212yuyv) を参照</a>	<a href="#">NV12/ NV21 to UYVY/ YUYV (nv122uyvy、nv122yuyv、nv212uyvy、nv212yuyv) を参照</a>	<a href="#">NV21 to YUV4 (nv212yuv4) を参照</a>	<a href="#">NV12/ NV21 to RGB/BGR (nv122rgb、nv122bgr、nv212rgb、nv212bgr) を参照</a>	<a href="#">NV12/ NV21 to RGB/BGR (nv122rgb、nv122bgr、nv212rgb、nv212bgr) を参照</a>
IYUV	<a href="#">IYUV to RGBA/RGB (iyuv2rgba、iyuv2rgb) を参照</a>	<a href="#">IYUV to NV12 (iyuv2nv12) を参照</a>		なし			<a href="#">IYUV to YUV4 (iyuv2yuv4) を参照</a>	<a href="#">IYUV to RGBA/RGB (iyuv2rgba、iyuv2rgb) を参照</a>	

UYVY	UYVY to RGBA (uyvy2rgba) を参照	UYVY to NV12 (uyvy2nv12) を参照		UYVY to IYUV (uyvy2iyuv) を参照	なし				
YUYV	YUYV to RGBA (yuyv2rgba) を参照	YUYV to NV12 (yuyv2nv12) を参照		YUYV to IYUV (yuyv2iyuv) を参照		なし			
YUV4							なし		
RGB		RGB/BGR to NV12/ NV21 (rgb2nv12 、 bgr2nv12、 rgb2nv21、 bgr2nv21) を参照	RGB/BGR to NV12/ NV21 (rgb2nv12 、 bgr2nv12、 rgb2nv21、 bgr2nv21) を参照	RGBA/RGB to IYUV (rgba2iyuv 、 rgb2iyuv) を参照	RGB/BGR to UYVY/ YUYV (rgb2uyvy 、 rgb2yuyv、 bgr2uyvy、 bgr2yuyv) を参照	RGB/BGR to UYVY/ YUYV (rgb2uyvy 、 rgb2yuyv、 bgr2uyvy、 bgr2yuyv) を参照	RGBA/RGB to YUV4 (rgba2yuv 4、 rgb2yuv4) を参照		BGR to RGB、RGB to BGR (bgr2rgb、 rgb2bgr) を参照
BGR		RGB/BGR to NV12/ NV21 (rgb2nv12 、 bgr2nv12、 rgb2nv21、 bgr2nv21) を参照	RGB/BGR to NV12/ NV21 (rgb2nv12 、 bgr2nv12、 rgb2nv21、 bgr2nv21) を参照		RGB/BGR to UYVY/ YUYV (rgb2uyvy 、 rgb2yuyv、 bgr2uyvy、 bgr2yuyv) を参照	RGB/BGR to UYVY/ YUYV (rgb2uyvy 、 rgb2yuyv、 bgr2uyvy、 bgr2yuyv) を参照		BGR to RGB、RGB to BGR (bgr2rgb、 rgb2bgr) を参照	

### その他の色変換

上記以外に、BGR/RGB <-> HSV、BGR/RGB <-> HLS、BGR/RGB <-> YCrCb、BGR/RGB <-> XYZ、および RGB <-> BGR 変換が追加されています。

### RGB から YUV への変換行列

次に、RGB データから YUV データへの変換式を示します。

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

### YUV から RGB への変換行列

次に、YUV データから RGB データへの変換式を示します。

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

参照: <http://www.fourcc.org/fccyvrgb.php>

## RGBA/RGB to YUV4 (rgba2yuv4、rgb2yuv4)

rgba2yuv4 は 4 チャンネルの RGBA 画像を YUV444 フォーマットに変換し、rgb2yuv4 は 3 チャンネル RGB 画像を YUV444 フォーマットに変換します。出力は Y、U、および V ストリームです。

## API 構文

```
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)
```

## パラメーターの説明

次の表に、テンプレートと関数のパラメーターを説明します。

表 96: rgba2yuv4 および rgb2yuv4 関数のパラメーターの説明

パラメーター	説明
SRC_T	入力ピクセルのデータ型。8 ビット、符号なし、4 (RGBA) および 3 (RGB) チャンネル (XF_8UC4 および XF_8UC3) のみサポート。
DST_T	出力ピクセルのデータ型。8 ビット、符号なし、1 チャンネル (XF_8UC1) のみサポート。
ROWS	入力および出力画像の最大高さ。
COLS	入力および出力画像の最大幅。8 ピクセル モードの場合は 8 の倍数で指定。
NPC	1 サイクルごとに処理されるピクセル数。1 ピクセルの場合は XF_NPPC1、8 ピクセルの場合は XF_NPPC8。
_src	サイズ (ROWS, COLS) の入力 Y プレーン。
_y_image	サイズ (ROWS, COLS) の出力 Y 画像。
_u_image	サイズ (ROWS, COLS) の出力 U 画像。
_v_image	サイズ (ROWS, COLS) の出力 V 画像。

## リソース使用量

次の表に、Xczu9eg-ffvb1156-1-i-es1 FPGA で HD (1080x1920) 画像を処理するために、Vivado HLS 2019.1 ツールを使用して生成された異なる設定での RGBA/RGB to YUV4 のリソース使用量を示します。

表 97: rgba2yuv4 および rgb2yuv4 関数のリソース使用量のサマリ

動作モード	動作周波数 (MHz)	使用量の見積もり				
		BRAM_18K	DSP_48E	FF	LUT	CLB
1 ピクセル	300	0	9	589	328	96

## パフォーマンス見積もり

次の表に、Xczu9eg-ffvb1156-1-i-es1 でグレースケール HD (1080x1920) 画像を処理するために、Vivado HLS 2019.1 ツールで生成された異なる設定での RGBA/RGB to YUV4 のパフォーマンス見積もりを示します。

表 98: rgba2yuv4 および rgb2yuv4 関数のパフォーマンス見積りのサマリ

動作モード	レイテンシ見積り
	最大レイテンシ (ms)
1 ピクセル動作 (300 MHz)	1.89

## RGBA/RGB to IYUV (rgba2iyuv、rgb2iyuv)

rgba2iyuv 関数は 4 チャンネルの RGBA 画像を IYUV (4:2:0) フォーマットに変換し、rgb2iyuv 関数は 3 チャンネル RGB 画像を IYUV (4:2:0) フォーマットに変換します。出力は Y、U、および V プレーンです。IYUV にはサブサンプリングされたデータが格納されるので、Y は RGBA/RGB の 1 ピクセルごとにサンプリングされ、U と V は 2 行および 2 列 (2x2) のピクセルごとにサンプリングされます。U および V プレーンのサイズは (rows/2)\*(columns/2) であり、連続する行を 1 つの行にカスケード接続することにより、プレーンのサイズが (rows/4)\*columns になります。

### API 構文

```
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)
```

### パラメーターの説明

次の表に、テンプレートと関数のパラメーターを説明します。

表 99: rgba2iyuv および rgb2iyuv 関数のパラメーターの説明

パラメーター	説明
SRC_T	入力ピクセルのデータ型。8 ビット、符号なし、4 (RGBA) および 3 (RGB) チャンネル (XF_8UC4 および XF_8UC3) のみサポート。
DST_T	出力ピクセルのデータ型。8 ビット、符号なし、1 チャンネル (XF_8UC1) のみサポート。
ROWS	入力および出力画像の最大高さ。
COLS	入力および出力画像の最大幅。8 ピクセル モードの場合は 8 の倍数で指定。
NPC	1 サイクルごとに処理されるピクセル数。1 ピクセルの場合は XF_NPPC1、8 ピクセルの場合は XF_NPPC8。
_src	サイズ (ROWS, COLS) の入力 Y プレーン。
_y_image	サイズ (ROWS, COLS) の出力 Y 画像。
_u_image	サイズ (ROWS/4, COLS) の出力 U 画像。
_v_image	サイズ (ROWS/4, COLS) の出力 V 画像。

### リソース使用量

次の表に、Xczu9eg-ffvb1156-1-i-es1 FPGA で HD (1080x1920) 画像を処理するために、Vivado HLS 2019.1 ツールを使用して生成された異なる設定での RGBA/RGB to IYUV のリソース使用量を示します。

表 100: rgba2iyuv および rgb2iyuv 関数のリソース使用量のサマリ

動作モード	動作周波数 (MHz)	使用量の見積もり				
		BRAM_18K	DSP_48E	FF	LUT	CLB
1 ピクセル	300	0	9	816	472	149

### パフォーマンス見積もり

次の表に、Xczu9eg-ffvb1156-1-i-es1 でグレースケール HD (1080x1920) 画像を処理するために、Vivado HLS 2019.1 ツールを使用して生成された異なる設定での RGBA/RGB to IYUV のパフォーマンス見積もりを示します。

表 101: rgba2iyuv および rgb2iyuv 関数のパフォーマンス見積もりのサマリ

動作モード	レイテンシ見積もり
	最大レイテンシ (ms)
1 ピクセル動作 (300 MHz)	1.8

## RGBA to NV12 (rgba2nv12)

rgba2nv12 関数は、4 チャンネルの RGBA 画像を NV12 (4:2:0) フォーマットに変換します。出力は Y プレーンおよびインターリーブされた UV プレーンです。NV12 にはサブサンプリングされたデータが格納されるので、Y は RGBA の 1 ピクセルごとにサンプリングされ、U および V は 2 行と 2 列 (2x2) のピクセルごとにサンプリングされます。UV プレーンのサイズは (rows/2)\*(columns/2) で、U と V の値がインターリーブされます。

### API 構文

```
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)
```

### パラメーターの説明

次の表に、テンプレートと関数のパラメーターを説明します。

表 102: rgba2nv12 関数のパラメーターの説明

パラメーター	説明
SRC_T	入力ピクセルのデータ型。8 ビット、符号なし、4 チャンネル (XF_8UC4) のみサポート。
Y_T	出力ピクセルのデータ型。8 ビット、符号なし、1 チャンネル (XF_8UC1) のみサポート。
UV_T	出力ピクセルのデータ型。8 ビット、符号なし、2 チャンネル (XF_8UC2) のみサポート。
ROWS	入力および出力画像の最大高さ。
COLS	入力および出力画像の最大幅。8 ピクセル モードの場合は 8 の倍数で指定。
NPC	1 サイクルごとに処理されるピクセル数。1 ピクセルの場合は XF_NPPC1、8 ピクセルの場合は XF_NPPC8。
_src	サイズ (ROWS, COLS) の入力 RGBA 画像。
_y	サイズ (ROWS, COLS) の出力 Y 画像。
_uv	サイズ (ROWS/2, COLS/2) の出力 UV 画像。



## リソース使用量

次の表に、Xczu9eg-ffvb1156-1-i-es1 FPGA で HD (1080x1920) 画像を処理するために、Vivado HLS 2019.1 ツールを使用して生成された異なる設定での RGBA to NV12 のリソース使用量を示します。

表 103: rgba2nv12 関数のリソース使用量のサマリ

動作モード	動作周波数 (MHz)	使用量の見積もり				
		BRAM_18K	DSP_48E	FF	LUT	CLB
1 ピクセル	300	0	9	802	452	128

## パフォーマンス見積もり

次の表に、Xczu9eg-ffvb1156-1-i-es1 でグレースケール HD (1080x1920) 画像を処理するために、Vivado HLS 2019.1 ツールを使用して生成された異なる設定での RGBA to NV12 のパフォーマンス見積もりを示します。

表 104: rgba2nv12 関数のパフォーマンス見積もりのサマリ

動作モード	レイテンシ見積もり
	最大レイテンシ (ms)
1 ピクセル動作 (300 MHz)	1.8

## RGBA to NV21 (rgba2nv21)

rgba2nv21 関数は、4 チャンルの RGBA 画像を NV21 (4:2:0) フォーマットに変換します。出力は Y プレーンおよびインターリーブされた VU プレーンです。NV21 にはサブサンプリングされたデータが格納されるので、Y は RGBA ピクセルごとにサンプリングされ、U および V は 2 行と 2 列 (2x2) の RGBA ピクセルごとにサンプリングされます。UV プレーンのサイズは (rows/2)\*(columns/2) で、V と U の値がインターリーブされます。

## API 構文

```
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)
```

## パラメーターの説明

次の表に、テンプレートと関数のパラメーターを説明します。

表 105: rgba2nv21 関数のパラメーターの説明

パラメーター	説明
SRC_T	入力ピクセルのデータ型。8 ビット、符号なし、4 チャンル (XF_8UC4) のみサポート。
Y_T	出力ピクセルのデータ型。8 ビット、符号なし、1 チャンル (XF_8UC1) のみサポート。
UV_T	出力ピクセルのデータ型。8 ビット、符号なし、1 チャンル (XF_8UC2) のみサポート。
ROWS	入力および出力画像の最大高さ。
COLS	入力および出力画像の最大幅。8 ピクセル モードの場合は 8 の倍数で指定。
NPC	1 サイクルごとに処理されるピクセル数。1 ピクセルの場合は XF_NPPC1、8 ピクセルの場合は XF_NPPC8。

表 105: rgba2nv21 関数のパラメーターの説明 (続き)

パラメーター	説明
_src	サイズ (ROWS, COLS) の入力 RGBA 画像。
_y	サイズ (ROWS, COLS) の出力 Y 画像。
_uv	サイズ (ROWS/2, COLS/2) の出力 UV 画像。

## リソース使用量

次の表に、Xczu9eg-ffvb1156-1-i-es1 FPGA で HD (1080x1920) 画像を処理するために、Vivado HLS 2019.1 ツールを使用して生成された異なる設定での RGBA to NV21 のリソース使用量を示します。

表 106: rgba2nv21 関数のリソース使用量のサマリ

動作モード	動作周波数 (MHz)	使用量の見積もり				
		BRAM_18K	DSP_48E	FF	LUT	CLB
1 ピクセル	300	0	9	802	453	131

## パフォーマンス見積もり

次の表に、Xczu9eg-ffvb1156-1-i-es1 でグレースケール HD (1080x1920) 画像を処理するために、Vivado HLS 2019.1 ツールを使用して生成された異なる設定での RGBA to NV21 のパフォーマンス見積もりを示します。

表 107: rgba2nv21 関数のパフォーマンス見積もりのサマリ

動作モード	レイテンシ見積もり
	最大レイテンシ (ms)
1 ピクセル動作 (300 MHz)	1.89

## YUYV to RGBA (yuyv2rgba)

yuyv2rgba 関数は、1 チャンルの YUYV (YUV 4:2:2) 画像フォーマットを 4 チャンルの RGBA 画像に変換します。YUYV はサブサンプリングフォーマットで、RGBA が 2 つ得られます。YUYV は 16 ビット値、RGBA は 32 ビット値です。

## API 構文

```
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)
```

## パラメーターの説明

次の表に、テンプレートと関数のパラメーターを説明します。

表 108: yuyv2rgba 関数のパラメーターの説明

パラメーター	説明
SRC_T	入力ピクセルのデータ型。16 ビット、符号なし、1 チャンネル (XF_16UC1) のみサポート。

表 108: yuyv2rgba 関数のパラメーターの説明 (続き)

パラメーター	説明
DST_T	出力ピクセルのデータ型。8 ビット、符号なし、4 チャンネル (XF_8UC4) のみサポート。
ROWS	入力および出力画像の最大高さ。
COLS	入力および出力画像の最大幅。8 ピクセル モードの場合は 8 の倍数で指定。
NPC	1 サイクルごとに処理されるピクセル数。1 ピクセルの場合は XF_NPPC1、8 ピクセルの場合は XF_NPPC8。
_src	サイズ (ROWS, COLS) の入力画像。
_dst	サイズ (ROWS, COLS) の出力画像。

## リソース使用量

次の表に、Xczu9eg-ffvb1156-1-i-es1 FPGA で HD (1080x1920) 画像を処理するために、Vivado HLS 2019.1 ツールを使用して生成された異なる設定での YUYV to RGBA のリソース使用量を示します。

表 109: yuyv2rgba 関数のリソース使用量のサマリ

動作モード	動作周波数 (MHz)	使用量の見積もり				
		BRAM_18K	DSP_48E	FF	LUT	CLB
1 ピクセル	300	0	6	765	705	165

## パフォーマンス見積もり

次の表に、Xczu9eg-ffvb1156-1-i-es1 でグレースケール HD (1080x1920) 画像を処理するために、Vivado HLS 2019.1 ツールで生成された異なる設定での YUYV to RGBA のパフォーマンス見積もりを示します。

表 110: yuyv2rgba 関数のパフォーマンス見積もりのサマリ

動作モード	レイテンシ見積もり
	最大レイテンシ (ms)
1 ピクセル動作 (300 MHz)	6.9

## YUYV to NV12 (yuyv2nv12)

yuyv2nv12 関数は、1 チャンネルの YUYV (YUV 4:2:2) 画像フォーマットを NV12 (YUV 4:2:0) フォーマットに変換します。YUYV はサブサンプリング フォーマットで、YUYV 値の 1 セットから Y 値が 2 つと U 値および V 値が 1 つずつ得られます。

## API 構文

```
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)
```

## パラメーターの説明

次の表に、テンプレートと関数のパラメーターを説明します。

表 111: yuyv2nv12 関数のパラメーターの説明

パラメーター	説明
SRC_T	入力ピクセルのデータ型。16 ビット、符号なし、1 チャンネル (XF_16UC1) のみサポート。
Y_T	出力ピクセルのデータ型。8 ビット、符号なし、1 チャンネル (XF_8UC1) のみサポート。
UV_T	出力 UV 画像ピクセルのデータ型。8 ビット、符号なし、2 チャンネル (XF_8UC2) のみサポート。
ROWS	入力および出力画像の最大高さ。
COLS	入力および出力画像の最大幅。8 ピクセル モードの場合は 8 の倍数で指定。
NPC	1 サイクルごとに処理されるピクセル数。1 ピクセルの場合は XF_NPPC1、8 ピクセルの場合は XF_NPPC8。
NPC_UV	1 サイクルごとに処理される UV 画像ピクセル数。1 ピクセルの操作の場合は XF_NPPC1、8 ピクセルの操作の場合は XF_NPPC8。
_src	サイズ (ROWS, COLS) の入力画像。
_y_image	サイズ (ROWS, COLS) 出力 Y プレーン。
_uv_image	サイズ (ROWS/2, COLS/2) の出力 U プレーン。

## リソース使用量

次の表に、Xczu9eg-ffvb1156-1-i-es1 FPGA で HD (1080x1920) 画像を処理するために、Vivado HLS 2019.1 ツールを使用して生成された異なる設定での YUYV to NV12 のリソース使用量を示します。

表 112: yuyv2nv12 関数のリソース使用量のサマリ

動作モード	動作周波数 (MHz)	使用量の見積もり				
		BRAM_18K	DSP_48E	FF	LUT	CLB
1 ピクセル	300	0	0	831	491	149
8 ピクセル	150	0	0	1196	632	161

## パフォーマンス見積もり

次の表に、Xczu9eg-ffvb1156-1-i-es1 でグレースケール HD (1080x1920) 画像を処理するために、Vivado HLS 2019.1 ツールで生成された異なる設定での YUYV to NV12 のパフォーマンス見積もりを示します。

表 113: yuyv2nv12 関数のパフォーマンス見積もりのサマリ

動作モード	レイテンシ見積もり
	最大レイテンシ (ms)
1 ピクセル動作 (300 MHz)	6.9
8 ピクセル動作 (150 MHz)	1.7

## YUYV to IYUV (yuyv2iyuv)

yuyv2iyuv は、1 チャンネルの YUYV (YUV 4:2:2) 画像フォーマットを IYUV (4:2:0) フォーマットに変換します。関数の出力は、Y、U、V プレーンです。YUYV はサブサンプリングフォーマットで、YUYV 値の 1 セットから Y 値が 2 つと U 値および V 値が 1 つずつ得られます。U および V 値は IYUV (4:2:0) フォーマットで 2 行と 2 列 (2x2) ごとにサンプリングされるので、奇数行の U および V 値は破棄されます。

## API 構文

```
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)
```

## パラメーターの説明

次の表に、テンプレートと関数のパラメーターを説明します。

表 114: yuyv2iyuv 関数のパラメーターの説明

パラメーター	説明
SRC_T	入力ピクセルのデータ型。16 ビット、符号なし、1 チャンネル (XF_16UC1) のみサポート。
DST_T	出力ピクセルのデータ型。8 ビット、符号なし、1 チャンネル (XF_8UC1) のみサポート。
ROWS	入力および出力画像の最大高さ。
COLS	入力および出力画像の最大幅。8 ピクセル モードの場合は 8 の倍数で指定。
NPC	1 サイクルごとに処理されるピクセル数。1 ピクセルの場合は XF_NPPC1、8 ピクセルの場合は XF_NPPC8。
_src	サイズ (ROWS, COLS) の入力画像。
_y_image	サイズ (ROWS, COLS) 出力 Y プレーン。
_u_image	サイズ (ROWS/4, COLS) の出力 U プレーン。
_v_image	サイズ (ROWS/4, COLS) の出力 V プレーン。

## リソース使用量

次の表に、ザイリンクス Xczu9eg-ffvb1156-1-i-es1 FPGA で HD (1080x1920) 画像を処理するために、Vivado HLS 2019.1 ツールを使用して生成された異なる設定での YUYV to IYUV のリソース使用量を示します。

表 115: yuyv2iyuv 関数のリソース使用量のサマリ

動作モード	動作周波数 (MHz)	使用量の見積もり				
		BRAM_18K	DSP_48E	FF	LUT	CLB
1 ピクセル	300	0	0	835	497	149
8 ピクセル	150	0	0	1428	735	210

## パフォーマンス見積もり

次の表に、ザイリンクス Xczu9eg-ffvb1156-1-i-es1 でグレースケール HD (1080x1920) 画像を処理するために、Vivado HLS 2019.1 ツールを使用して生成された異なる設定での YUYV to IYUV のパフォーマンス見積もりを示します。

表 116: yuyv2iyuv 関数のパフォーマンス見積もり

動作モード	レイテンシ見積もり
	最大レイテンシ (ms)
1 ピクセル動作 (300 MHz)	6.9
8 ピクセル動作 (150 MHz)	1.7

## UYVY to IYUV (uyvy2iyuv)

uyvy2iyuv 関数は、UYVY (YUV 4:2:2) の 1 チャンネル画像を IYUV フォーマットに変換します。出力は Y、U、V プレーンです。UYVY はサブサンプリング フォーマットです。UYVY 値の 1 セットから Y 値が 2 つと U 値および V 値が 1 つずつ得られます。

### API 構文

```
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)
```

### パラメーターの説明

次の表に、テンプレートと関数のパラメーターを説明します。

表 117: uyvy2iyuv 関数のパラメーターの説明

パラメーター	説明
SRC_T	入力ピクセルのデータ型。16 ビット、符号なし、1 チャンネル (XF_16UC1) のみサポート。
DST_T	出力ピクセルのデータ型。8 ビット、符号なし、1 チャンネル (XF_8UC1) のみサポート。
ROWS	入力および出力画像の最大高さ。
COLS	入力および出力画像の最大幅。8 ピクセル モードの場合は 8 の倍数で指定。
NPC	1 サイクルごとに処理されるピクセル数。1 ピクセルの場合は XF_NPPC1、8 ピクセルの場合は XF_NPPC8。
_src	サイズ (ROWS, COLS) の入力画像。
_y_image	サイズ (ROWS, COLS) 出力 Y プレーン。
_u_image	サイズ (ROWS/4, COLS) の出力 U プレーン。
_v_image	サイズ (ROWS/4, COLS) の出力 V プレーン。

### リソース使用量

次の表に、Xczu9eg-ffvb1156-1-i-es1 FPGA で HD (1080x1920) 画像を処理するために、Vivado HLS 2019.1 ツールを使用して生成された異なる設定での UYVY to IYUV のリソース使用量を示します。

表 118: uyvy2iyuv 関数のリソース使用量のサマリ

動作モード	動作周波数 (MHz)	使用量の見積もり				
		BRAM_18K	DSP_48E	FF	LUT	CLB
1 ピクセル	300	0	0	835	494	139
8 ピクセル	150	0	0	1428	740	209

### パフォーマンス見積もり

次の表に、Xczu9eg-ffvb1156-1-i-es1 でグレースケール HD (1080x1920) 画像を処理するために、Vivado HLS 2019.1 ツールを使用して生成された異なる設定での UYVY to IYUV のパフォーマンス見積もりを示します。

表 119: uyvy2iyuv 関数のパフォーマンス見積もりのサマリ

動作モード	レイテンシ見積もり
	最大レイテンシ (ms)
1 ピクセル動作 (300 MHz)	6.9
8 ピクセル動作 (150 MHz)	1.7

## UYVY to RGBA (uyvy2rgba)

uyvy2rgba 関数は、UYVY (YUV 4:2:2) の 1 チャンネル画像を 4 チャンネルの RGBA 画像に変換します。UYVY はサブサンプリングフォーマットで、UYVY 値の 1 セット値から 2 つの RGBA ピクセル値が得られます。UYVY は 16 ビット値、RGBA は 32 ビット値です。

### API 構文

```
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)
```

### パラメーターの説明

次の表に、テンプレートと関数のパラメーターを説明します。

表 120: uyvy2rgba 関数のパラメーターの説明

パラメーター	説明
SRC_T	入力ピクセルのデータ型。16 ビット、符号なし、1 チャンネル (XF_16UC1) のみサポート。
DST_T	出力ピクセルのデータ型。8 ビット、符号なし、1 チャンネル (XF_8UC1) のみサポート。
ROWS	入力および出力画像の最大高さ。
COLS	入力および出力画像の最大幅。8 ピクセル モードの場合は 8 の倍数で指定。
NPC	1 サイクルごとに処理されるピクセル数。1 ピクセルの場合は XF_NPPC1、8 ピクセルの場合は XF_NPPC8。
_src	サイズ (ROWS, COLS) の入力画像。
_dst	サイズ (ROWS, COLS) の出力画像。

### リソース使用量

次の表に、Xczu9eg-ffvb1156-1-i-es1 FPGA で HD (1080x1920) 画像を処理するために、Vivado HLS 2019.1 ツールを使用して生成された異なる設定での UYVY to RGBA のリソース使用量を示します。

表 121: uyvy2rgba 関数のリソース使用量のサマリ

動作モード	動作周波数 (MHz)	使用量の見積もり				
		BRAM_18K	DSP_48E	FF	LUT	CLB
1 ピクセル	300	0	6	773	704	160

## パフォーマンス見積もり

次の表に、Xczu9eg-ffvb1156-1-i-es1 でグレースケール HD (1080x1920) 画像を処理するために、Vivado HLS 2019.1 ツールで生成された異なる設定での UYVY to RGBA のパフォーマンス見積もりを示します。

表 122: uyvy2rgba 関数のパフォーマンス見積もりのサマリ

動作モード	レイテンシ見積もり
	最大レイテンシ (ms)
1 ピクセル動作 (300 MHz)	6.8

## UYVY to NV12 (uyvy2nv12)

uyvy2nv12 関数は、UYVY (YUV 4:2:2) の 1 チャンネル画像を NV12 フォーマットに変換します。出力は Y および UV プレーンです。UYVY はサブサンプリング フォーマットで、1 つの UYVY のセット値から Y 値が 2 つと U 値および V 値が 1 つずつ得られます。

### API 構文

```
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)
```

### パラメーターの説明

次の表に、テンプレートと関数のパラメーターを説明します。

表 123: uyvy2nv12 関数のパラメーターの説明

パラメーター	説明
SRC_T	入力ピクセルのデータ型。16 ビット、符号なし、1 チャンネル (XF_16UC1) のみサポート。
Y_T	出力ピクセルのデータ型。8 ビット、符号なし、1 チャンネル (XF_8UC1) のみサポート。
UV_T	出力 UV 画像ピクセルのデータ型。8 ビット、符号なし、2 チャンネル (XF_8UC2) のみサポート。
ROWS	入力および出力画像の最大高さ。
COLS	入力および出力画像の最大幅。8 ピクセル モードの場合は 8 の倍数で指定。
NPC	1 サイクルごとに処理されるピクセル数。1 ピクセルの場合は XF_NPPC1、8 ピクセルの場合は XF_NPPC8。
NPC_UV	1 サイクルごとに処理される UV 画像ピクセル数。1 ピクセルの操作の場合は XF_NPPC1、8 ピクセルの操作の場合は XF_NPPC4。
_src	サイズ (ROWS, COLS) の入力画像。
_y_image	サイズ (ROWS, COLS) 出力 Y プレーン。
_uv_image	サイズ (ROWS/2, COLS/2) の出力 U プレーン。

### リソース使用量

次の表に、Xczu9eg-ffvb1156-1-i-es1 FPGA で HD (1080x1920) 画像を処理するために、Vivado HLS 2019.1 ツールを使用して生成された異なる設定での UYVY to NV12 のリソース使用量を示します。



表 124: uyvy2nv12 関数のリソース使用量のサマリ

動作モード	動作周波数 (MHz)	使用量の見積もり				
		BRAM_18K	DSP_48E	FF	LUT	CLB
1 ピクセル	300	0	0	831	488	131
8 ピクセル	150	0	0	1235	677	168

### パフォーマンス見積もり

次の表に、Xczu9eg-ffvb1156-1-i-es1 でグレースケール HD (1080x1920) 画像を処理するために、Vivado HLS 2019.1 ツールで生成された異なる設定での UYVY to NV12 のパフォーマンス見積もりを示します。

表 125: uyvy2nv12 関数のパフォーマンス見積もりのサマリ

動作モード	レイテンシ見積もり
	最大レイテンシ (ms)
1 ピクセル動作 (300 MHz)	6.9
8 ピクセル動作 (150 MHz)	1.7

## IYUV to RGBA/RGB (iyuv2rgba、iyuv2rgb)

`iyuv2rgba` 関数は、1 チャンネルの IYUV (YUV 4:2:0) 画像を 4 チャンネルの RGBA 画像に変換します。`iyuv2rgb` 関数は、1 チャンネルの IYUV (YUV 4:2:0) 画像を 3 チャンネルの RGB 画像に変換します。入力 Y、U、V プレーンです。IYUV はサブサンプリングフォーマットで、U および V 値は RGBA/RGB ピクセルの 2 行および 2 列に対して 1 回サンプリングされます。サイズ (columns/2) の連続行のデータを組み合わせることにより、サイズ (columns) の 1 行が形成されます。

### API 構文

```
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)
```

### パラメーターの説明

次の表に、テンプレートと関数のパラメーターを説明します。

表 126: iyuv2rgba および iyuv2rgb 関数のパラメーターの説明

パラメーター	説明
SRC_T	入力ピクセルのデータ型。8 ビット、符号なし、1 チャンネル (XF_8UC1) のみサポート。
DST_T	出力ピクセルのデータ型。8 ビット、符号なし、4 (RGBA) および 3 (RGB) チャンネル (XF_8UC4 および XF_8UC3) のみサポート。

表 126: iyuv2rgba および iyuv2rgb 関数のパラメーターの説明 (続き)

パラメーター	説明
ROWS	入力および出力画像の最大高さ。
COLS	入力および出力画像の最大幅。8 ピクセル モードの場合は 8 の倍数で指定。
NPC	1 サイクルごとに処理されるピクセル数。1 ピクセルの場合は XF_NPPC1、8 ピクセルの場合は XF_NPPC8。
src_y	サイズ (ROWS, COLS) の入力 Y プレーン。
src_u	サイズ (ROWS/4, COLS) の入力 U プレーン。
src_v	サイズ (ROWS/4, COLS) の入力 V プレーン。
_dst0	サイズ (ROWS, COLS) の出力 RGBA 画像。

## リソース使用量

次の表に、Xczu9eg-ffvb1156-1-i-es1 FPGA で HD (1080x1920) 画像を処理するために、Vivado HLS 2019.1 ツールを使用して生成された異なる設定での IYUV to RGBA/RGB のリソース使用量を示します。

表 127: iyuv2rgba および iyuv2rgb 関数のリソース使用量のサマリ

動作モード	動作周波数 (MHz)	使用量の見積もり				
		BRAM_18K	DSP_48E	FF	LUT	CLB
1 ピクセル	300	2	5	1208	728	196

## パフォーマンス見積もり

次の表に、Xczu9eg-ffvb1156-1-i-es1 でグレースケール HD (1080x1920) 画像を処理するために、Vivado HLS 2019.1 ツールで生成された異なる設定での IYUV to RGBA/RGB のパフォーマンス見積もりを示します。

表 128: iyuv2rgba および iyuv2rgb 関数のパフォーマンス見積もりのサマリ

動作モード	レイテンシ見積もり
	最大レイテンシ (ms)
1 ピクセル動作 (300 MHz)	6.9

## IYUV to NV12 (iyuv2nv12)

iyuv2nv12 関数は、1 チャンネル IYUV 画像を NV12 フォーマットに変換します。入力は U および V プレーンです。Y プレーンはどちらのフォーマットでも同じなので、Y プレーンを処理する必要はありません。U 値および V 値は、プレーンインターリーブからピクセルインターリーブに変換されます。

## API 構文

```
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)
```

## パラメーターの説明

次の表に、テンプレートと関数のパラメーターを説明します。

表 129: iyuv2nv12 関数のパラメーターの説明

パラメーター	説明
SRC_T	入力ピクセルのデータ型。8 ビット、符号なし、1 チャンネル (XF_8UC1) のみサポート。
UV_T	出力ピクセルのデータ型。8 ビット、符号なし、2 チャンネル (XF_8UC2) のみサポート。
ROWS	入力および出力画像の最大高さ。
COLS	入力および出力画像の最大幅。8 ピクセル モードの場合は 8 の倍数で指定。
NPC	1 サイクルごとに処理されるピクセル数。1 ピクセルの場合は XF_NPPC1、8 ピクセルの場合は XF_NPPC8。
NPC_UV	1 サイクルごとに処理される UV ピクセル数。1 ピクセル動作の場合は XF_NPPC1、4 ピクセル動作の場合は XF_NPPC4。
src_y	サイズ (ROWS, COLS) の入力 Y プレーン。
src_u	サイズ (ROWS/4, COLS) の入力 U プレーン。
src_v	サイズ (ROWS/4, COLS) の入力 V プレーン。
_y_image	サイズ (ROWS, COLS) の出力 V プレーン。
_uv_image	サイズ (ROWS/2, COLS/2) の出力 UV プレーン。

## リソース使用量

次の表に、Xczu9eg-ffvb1156-1-i-es1 FPGA で HD (1080x1920) 画像を処理するために、Vivado HLS 2019.1 ツールを使用して生成された異なる設定での IYUV to NV12 のリソース使用量を示します。

表 130: iyuv2nv12 関数のリソース使用量のサマリ

動作モード	動作周波数 (MHz)	使用量の見積もり				
		BRAM_18K	DSP_48E	FF	LUT	CLB
1 ピクセル	300	0	12	907	677	158
8 ピクセル	150	0	12	1591	1022	235

## パフォーマンス見積もり

次の表に、Xczu9eg-ffvb1156-1-i-es1 でグレースケール HD (1080x1920) 画像を処理するために、Vivado HLS 2019.1 ツールを使用して生成された異なる設定での IYUV to NV12 のパフォーマンス見積もりを示します。

表 131: iyuv2nv12 関数のパフォーマンス見積もりのサマリ

動作モード	レイテンシ見積もり
	最大レイテンシ (ms)
1 ピクセル動作 (300 MHz)	6.9
8 ピクセル動作 (150 MHz)	1.7

## IYUV to YUV4 (iyuv2yuv4)

`iyuv2yuv4` 関数は、1 チャンネル IYUV 画像を YUV444 フォーマットに変換します。Y プレーンはどちらのフォーマットでも同じです。入力は IYUV 画像の U および V プレーンで、出力は YUV4 画像の U および V プレーンです。IYUV には、サブサンプリングされた U および V 値が格納されます。YUV フォーマットには、各ピクセルの U および V 値が格納されます。同じ U および V 値が 2 行および 2 列 (2x2) のピクセルに複製され、YUV444 フォーマットに必要なデータが取得されます。

### API 構文

```
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)
```

### パラメーターの説明

次の表に、テンプレートと関数のパラメーターを説明します。

表 132: `iyuv2yuv4` 関数のパラメーターの説明

パラメーター	説明
SRC_T	入力ピクセルのデータ型。8 ビット、符号なし、1 チャンネル (XF_8UC1) のみサポート。
ROWS	入力および出力画像の最大高さ。
COLS	入力および出力画像の最大幅。8 ピクセル モードの場合は 8 の倍数で指定。
NPC	1 サイクルごとに処理されるピクセル数。1 ピクセルの場合は XF_NPPC1、8 ピクセルの場合は XF_NPPC8。
src_y	サイズ (ROWS, COLS) の入力 Y プレーン。
src_u	サイズ (ROWS/4, COLS) の入力 U プレーン。
src_v	サイズ (ROWS/4, COLS) の入力 V プレーン。
_y_image	サイズ (ROWS, COLS) の出力 Y 画像。
_u_image	サイズ (ROWS, COLS) の出力 U 画像。
_v_image	サイズ (ROWS, COLS) の出力 V 画像。

### リソース使用量

次の表に、Xczu9eg-ffvb1156-1-i-es1 FPGA で HD (1080x1920) 画像を処理するために、Vivado HLS 2019.1 ツールを使用して生成された異なる設定での IYUV to YUV4 のリソース使用量を示します。

表 133: `iyuv2yuv4` 関数のリソース使用量のサマリ

動作モード	動作周波数 (MHz)	使用量の見積もり				
		BRAM_18K	DSP_48E	FF	LUT	CLB
1 ピクセル	300	0	0	1398	870	232
8 ピクセル	150	0	0	2134	1214	304

## パフォーマンス見積もり

次の表に、Xczu9eg-ffvb1156-1-i-es1 でグレースケール HD (1080x1920) 画像を処理するために、Vivado HLS 2019.1 ツールを使用して生成された異なる設定での IYUV to YUV4 のパフォーマンス見積もりを示します。

表 134: iyuv2yuv4 関数のパフォーマンス見積もりのサマリ

動作モード	レイテンシ見積もり
	最大レイテンシ (ms)
1 ピクセル動作 (300 MHz)	13.8
8 ピクセル動作 (150 MHz)	3.4

## NV12 to IYUV (nv12iyuv)

nv12iyuv 関数は、NV12 フォーマットを IYUV フォーマットに変換します。入力はインターリーブされた UV プレーンのみであり、出力は U および V プレーンです。Y プレーンはどちらのフォーマットでも同じなので、Y プレーン进行处理する必要はありません。U 値および V 値は、ピクセルインターリーブからプレーンインターリーブに変換されます。

## API 構文

```
template<int SRC_T, int UV_T, int ROWS, int COLS, int NPC=1, int NPC_UV=1>
void nv12iyuv(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)
```

## パラメーターの説明

次の表に、テンプレートと関数のパラメーターを説明します。

表 135: nv12iyuv 関数のパラメーターの説明

パラメーター	説明
SRC_T	入力ピクセルのデータ型。8 ビット、符号なし、1 チャンネル (XF_8UC1) のみサポート。
UV_T	入力ピクセルのデータ型。8 ビット、符号なし、2 チャンネル (XF_8UC2) のみサポート。
ROWS	入力および出力画像の最大高さ。
COLS	入力および出力画像の最大幅 (8 ピクセル モードの場合は 8 の倍数で指定)。
NPC	1 サイクルごとに処理されるピクセル数。1 ピクセルの場合は XF_NPPC1、8 ピクセルの場合は XF_NPPC8。
NPC_UV	1 サイクルごとに処理される UV 画像ピクセル数。1 ピクセルの操作の場合は XF_NPPC1、4 ピクセルの操作の場合は XF_NPPC4。
src_y	サイズ (ROWS, COLS) の入力 Y プレーン。
src_uv	サイズ (ROWS/2, COLS/2) の入力 UV プレーン。
_y_image	サイズ (ROWS, COLS) 出力 Y プレーン。
_u_image	サイズ (ROWS/4, COLS) の出力 U プレーン。
_v_image	サイズ (ROWS/4, COLS) の出力 V プレーン。

## リソース使用量

次の表に、Xczu9eg-ffvb1156-1-i-es1 FPGA で HD (1080x1920) 画像を処理するために、Vivado HLS 2019.1 ツールを使用して生成された異なる設定での NV12 to IYUV のリソース使用量を示します。

表 136: nv12iyuv 関数のリソース使用量のサマリ

動作モード	動作周波数 (MHz)	使用量の見積もり				
		BRAM_18K	DSP_48E	FF	LUT	CLB
1 ピクセル	300	0	1	1344	717	208
8 ピクセル	150	0	1	1961	1000	263

## パフォーマンス見積もり

次の表に、Xczu9eg-ffvb1156-1-i-es1 でグレースケール HD (1080x1920) 画像を処理するために、Vivado HLS 2019.1 ツールで生成された異なる設定での NV12 to IYUV のパフォーマンス見積もりを示します。

表 137: nv12iyuv 関数のパフォーマンス見積もりのサマリ

動作モード	レイテンシ見積もり
	最大レイテンシ (ms)
1 ピクセル動作 (300 MHz)	6.9
8 ピクセル動作 (150 MHz)	1.7

## NV12 to RGBA (nv12rgba)

nv12rgba 関数は、NV12 画像フォーマットを 4 チャンネル RGBA 画像に変換します。入力 は Y および UV プレーンです。NV12 にはサブサンプリングされたデータが格納されており、Y プレーンがユニット レートでサンプリングされ、2x2 の Y 値ごとに U 値と V 値が 1 つずつサンプリングされます。RGBA データを生成するため、各 U および V 値が 2x2 回複製されます。

## API 構文

```
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)
```

## パラメーターの説明

次の表に、テンプレートと関数のパラメーターを説明します。

表 138: nv122rgba 関数のパラメーターの説明

パラメーター	説明
SRC_T	入力ピクセルのデータ型。8 ビット、符号なし、1 チャンネル (XF_8UC1) のみサポート。
UV_T	入力ピクセルのデータ型。8 ビット、符号なし、2 チャンネル (XF_8UC2) のみサポート。
DST_T	出力ピクセルのデータ型。8 ビット、符号なし、4 チャンネル (XF_8UC4) のみサポート。
ROWS	入力および出力画像の最大高さ。

表 138: nv122rgba 関数のパラメーターの説明 (続き)

パラメーター	説明
COLS	入力および出力画像の最大幅。8 ピクセル モードの場合は 8 の倍数で指定。
NPC	1 サイクルごとに処理されるピクセル数。1 ピクセルの場合は XF_NPPC1、8 ピクセルの場合は XF_NPPC8。
src_y	サイズ (ROWS, COLS) の入力 Y プレーン。
src_uv	サイズ (ROWS/2, COLS/2) の入力 UV プレーン。
_dst0	サイズ (ROWS, COLS) の出力 RGBA 画像。

## リソース使用量

次の表に、Xczu9eg-ffvb1156-1-i-es1 FPGA で HD (1080x1920) 画像を処理するために、Vivado HLS 2019.1 ツールを使用して生成された異なる設定での NV12 to RGBA のリソース使用量を示します。

表 139: nv122rgba 関数のリソース使用量のサマリ

動作モード	動作周波数 (MHz)	使用量の見積もり				
		BRAM_18K	DSP_48E	FF	LUT	CLB
1 ピクセル	300	2	5	1191	708	195

## パフォーマンス見積もり

次の表に、Xczu9eg-ffvb1156-1-i-es1 でグレースケール HD (1080x1920) 画像を処理するために、Vivado HLS 2019.1 ツールで生成された異なる設定での NV12 to RGBA のパフォーマンス見積もりを示します。

表 140: nv122rgba 関数のパフォーマンス見積もりのサマリ

動作モード	レイテンシ見積もり
	最大レイテンシ (ms)
1 ピクセル動作 (300 MHz)	6.9

## NV12 to YUV4 (nv122yuv4)

nv122yuv4 関数は、NV12 画像フォーマットを YUV444 フォーマットに変換します。出力は U および V プレーンです。Y プレーンはどちらの画像フォーマットでも同じです。UV プレーンは 2x2 回複製され、YUV444 画像フォーマットの 1 つの U プレーンと V プレーンを表します。

## API 構文

```
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)
```

## パラメーターの説明

次の表に、テンプレートと関数のパラメーターを説明します。

表 141: nv122yuv4 関数のパラメーターの説明

パラメーター	説明
SRC_T	入力ピクセルのデータ型。8 ビット、符号なし、1 チャンネル (XF_8UC1) のみサポート。
UV_T	入力ピクセルのデータ型。8 ビット、符号なし、2 チャンネル (XF_8UC2) のみサポート。
ROWS	入力および出力画像の最大高さ。
COLS	入力および出力画像の最大幅 (8 ピクセル モードの場合は 8 の倍数で指定)。
NPC	1 サイクルごとに処理されるピクセル数。1 ピクセルの場合は XF_NPPC1、8 ピクセルの場合は XF_NPPC8。
NPC_UV	1 サイクルごとに処理される UV 画像ピクセル数。1 ピクセルの操作の場合は XF_NPPC1、4 ピクセルの操作の場合は XF_NPPC4。
src_y	サイズ (ROWS, COLS) の入力 Y プレーン。
src_uv	サイズ (ROWS/2, COLS/2) の入力 UV プレーン。
_y_image	サイズ (ROWS, COLS) 出力 Y プレーン。
_u_image	サイズ (ROWS, COLS) の出力 U プレーン。
_v_image	サイズ (ROWS, COLS) の出力 V プレーン。

## リソース使用量

次の表に、Xczu9eg-ffvb1156-1-i-es1 FPGA で HD (1080x1920) 画像を処理するために、Vivado HLS 2019.1 ツールを使用して生成された異なる設定での NV12 to YUV4 のリソース使用量を示します。

表 142: nv122yuv4 関数のリソース使用量のサマリ

動作モード	動作周波数 (MHz)	使用量の見積もり				
		BRAM_18K	DSP_48E	FF	LUT	CLB
1 ピクセル	300	0	0	1383	832	230
8 ピクセル	150	0	0	1772	1034	259

## パフォーマンス見積もり

次の表に、Xczu9eg-ffvb1156-1-i-es1 でグレースケール HD (1080x1920) 画像を処理するために、Vivado HLS 2019.1 ツールで生成された異なる設定での NV12 to YUV4 のパフォーマンス見積もりを示します。

表 143: nv122yuv4 関数のパフォーマンス見積もりのサマリ

動作モード	レイテンシ見積もり
	最大レイテンシ (ms)
1 ピクセル動作 (300 MHz)	13.8
8 ピクセル動作 (150 MHz)	3.4



## NV21 to IYUV (nv212iyuv)

nv212iyuv 関数は、NV21 画像フォーマットを IYUV 画像フォーマットに変換します。入力はインターリーブされた VU プレーンのみであり、出力は U および V プレーンです。Y プレーンはどちらのフォーマットでも同じなので、Y プレーンを処理する必要はありません。U 値および V 値は、ピクセル インターリーブからプレーン インターリーブに変換されます。

### API 構文

```
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)
```

### パラメーターの説明

次の表に、テンプレートと関数のパラメーターを説明します。

表 144: nv212iyuv 関数のパラメーターの説明

パラメーター	説明
SRC_T	入力ピクセルのデータ型。8 ビット、符号なし、1 チャンネル (XF_8UC1) のみサポート。
UV_T	入力ピクセルのデータ型。8 ビット、符号なし、2 チャンネル (XF_8UC2) のみサポート。
ROWS	入力および出力画像の最大高さ。
COLS	入力および出力画像の最大幅。8 ピクセル モードの場合は 8 の倍数で指定。
NPC	1 サイクルごとに処理されるピクセル数。1 ピクセルの場合は XF_NPPC1、8 ピクセルの場合は XF_NPPC8。
NPC_UV	1 サイクルごとに処理される UV 画像ピクセル数。1 ピクセルの操作の場合は XF_NPPC1、4 ピクセルの操作の場合は XF_NPPC4。
src_y	サイズ (ROWS, COLS) の入力 Y プレーン。
src_uv	サイズ (ROWS/2, COLS/2) の入力 UV プレーン。
_y_image	サイズ (ROWS, COLS) 出力 Y プレーン。
_u_image	サイズ (ROWS/4, COLS) の出力 U プレーン。
_v_image	サイズ (ROWS/4, COLS) の出力 V プレーン。

### リソース使用量

次の表に、Xczu9eg-ffvb1156-1-i-es1 FPGA で HD (1080x1920) 画像を処理するために、Vivado HLS 2019.1 ツールを使用して生成された異なる設定での NV21 to IYUV のリソース使用量を示します。

表 145: nv212iyuv 関数のリソース使用量のサマリ

動作モード	動作周波数 (MHz)	使用量の見積もり				
		BRAM_18K	DSP_48E	FF	LUT	CLB
1 ピクセル	300	0	1	1377	730	219
8 ピクセル	150	0	1	1975	1012	279

## パフォーマンス見積もり

次の表に、Xczu9eg-ffvb1156-1-i-es1 でグレースケール HD (1080x1920) 画像を処理するために、Vivado HLS 2019.1 ツールで生成された異なる設定での NV21 to IYUV のパフォーマンス見積もりを示します。

表 146: nv21iyuv 関数のパフォーマンス見積もりのサマリ

動作モード	レイテンシ見積もり
	最大レイテンシ (ms)
1 ピクセル動作 (300 MHz)	6.9
8 ピクセル動作 (150 MHz)	1.7

## NV21 to RGBA (nv21rgba)

nv21rgba 関数は、NV21 画像フォーマットを 4 チャンネル RGBA 画像に変換します。入力 Y および VU プレーンです。NV21 にはサブサンプリングされたデータが格納されており、Y プレーンがユニットレートでサンプリングされ、2x2 の Y 値ごとに U 値と V 値が 1 つずつサンプリングされます。RGBA データを生成するため、各 U および V 値が 2x2 回複製されます。

### API 構文

```
template<int SRC_T, int UV_T, int DST_T, int ROWS, int COLS, int NPC=1>
void nv21rgba(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)
```

### パラメーターの説明

次の表に、テンプレートと関数のパラメーターを説明します。

表 147: nv21rgba 関数のパラメーターの説明

パラメーター	説明
SRC_T	入力ピクセルのデータ型。8 ビット、符号なし、1 チャンネル (XF_8UC1) のみサポート。
UV_T	入力ピクセルのデータ型。8 ビット、符号なし、2 チャンネル (XF_8UC2) のみサポート。
DST_T	出力ピクセルのデータ型。8 ビット、符号なし、4 チャンネル (XF_8UC4) のみサポート。
ROWS	入力および出力画像の最大高さ。
COLS	入力および出力画像の最大幅。8 ピクセル モードの場合は 8 の倍数で指定。
NPC	1 サイクルごとに処理されるピクセル数。1 ピクセルの場合は XF_NPPC1、8 ピクセルの場合は XF_NPPC8。
src_y	サイズ (ROWS, COLS) の入力 Y プレーン。
src_uv	サイズ (ROWS/2, COLS/2) の入力 UV プレーン。
_dst0	サイズ (ROWS, COLS) の出力 RGBA 画像。

### リソース使用量

次の表に、Xczu9eg-ffvb1156-1-i-es1 FPGA で HD (1080x1920) 画像を処理するために、Vivado HLS 2019.1 ツールを使用して生成された異なる設定での NV21 to RGBA のリソース使用量を示します。

表 148: nv212rgba 関数のリソース使用量のサマリ

動作モード	動作周波数 (MHz)	使用量の見積もり				
		BRAM_18K	DSP_48E	FF	LUT	CLB
1 ピクセル	300	2	5	1170	673	183

### パフォーマンス見積もり

次の表に、Xczu9eg-ffvb1156-1-i-es1 でグレースケール HD (1080x1920) 画像を処理するために、Vivado HLS 2019.1 ツールで生成された異なる設定での NV12 to RGBA のパフォーマンス見積もりを示します。

表 149: nv212rgba 関数のパフォーマンス見積もりのサマリ

動作モード	レイテンシ見積もり
	最大レイテンシ (ms)
1 ピクセル動作 (300 MHz)	6.9

## NV21 to YUV4 (nv212yuv4)

nv212yuv4 関数は、NV12 画像フォーマットを YUV444 フォーマットに変換します。出力は U および V プレーンです。Y プレーンはどちらのフォーマットでも同じです。UV プレーンは 2x2 回複製され、YUV444 フォーマットの 1 つの U プレーンと V プレーンを表します。

### API 構文

```
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)
```

### パラメーターの説明

次の表に、テンプレートと関数のパラメーターを説明します。

表 150: nv212yuv4 関数のパラメーターの説明

パラメーター	説明
SRC_T	入力ピクセルのデータ型。8 ビット、符号なし、1 チャンネル (XF_8UC1) のみサポート。
UV_T	入力ピクセルのデータ型。8 ビット、符号なし、2 チャンネル (XF_8UC2) のみサポート。
ROWS	入力および出力画像の最大高さ。
COLS	入力および出力画像の最大幅 (8 ピクセル モードの場合は 8 の倍数で指定)。
NPC	1 サイクルごとに処理されるピクセル数。1 ピクセルの場合は XF_NPPC1、8 ピクセルの場合は XF_NPPC8。
NPC_UV	1 サイクルごとに処理される UV 画像ピクセル数。1 ピクセルの操作の場合は XF_NPPC1、4 ピクセルの操作の場合は XF_NPPC4。
src_y	サイズ (ROWS, COLS) の入力 Y プレーン。
src_uv	サイズ (ROWS/2, COLS/2) の入力 UV プレーン。

表 150: nv212yuv4 関数のパラメーターの説明 (続き)

パラメーター	説明
_y_image	サイズ (ROWS, COLS) 出力 Y プレーン。
_u_image	サイズ (ROWS, COLS) の出力 U プレーン。
_v_image	サイズ (ROWS, COLS) の出力 V プレーン。

## リソース使用量

次の表に、Xczu9eg-ffvb1156-1-i-es1 FPGA で HD (1080x1920) 画像を処理するために、Vivado HLS 2019.1 ツールを使用して生成された異なる設定での NV21 to YUV4 のリソース使用量を示します。

表 151: nv212yuv4 関数のリソース使用量のサマリ

動作モード	動作周波数 (MHz)	使用量の見積もり				
		BRAM_18K	DSP_48E	FF	LUT	CLB
1 ピクセル	300	0	0	1383	817	233
8 ピクセル	150	0	0	1887	1087	287

## パフォーマンス見積もり

次の表に、Xczu9eg-ffvb1156-1-i-es1 でグレースケール HD (1080x1920) 画像を処理するために、Vivado HLS 2019.1 ツールで生成された異なる設定での NV21 to YUV4 のパフォーマンス見積もりを示します。

表 152: nv212yuv4 関数のパフォーマンス見積もりのサマリ

動作モード	レイテンシ見積もり
	最大レイテンシ (ms)
1 ピクセル動作 (300 MHz)	13.8
8 ピクセル動作 (150 MHz)	3.5

## RGB to GRAY (rgb2gray)

rgb2gray 関数は、3 チャンネルの RGB 画像を GRAY フォーマットに変換します。

$$Y = 0.299 * R + 0.587 * G + 0.114 * B$$

説明:

- Y: グレー ピクセル
- R: 赤チャンネル
- G: 緑チャンネル
- B: 青チャンネル

## API 構文

```
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)
```

## パラメーターの説明

次の表に、テンプレートと関数のパラメーターを説明します。

表 153: RGB2GRAY 関数のパラメーターの説明

パラメーター	説明
SRC_T	入力ピクセルのデータ型。8 ビット、符号なし、3 チャンネル (XF_8UC3) のみサポート。
DST_T	出力ピクセルのデータ型。8 ビット、符号なし、1 チャンネル (XF_8UC1) のみサポート。
ROWS	入力および出力画像の最大高さ。
COLS	入力および出力画像の最大幅。
NPC	1 サイクルごとに処理されるピクセル数。
_src	RGB 入力画像
_dst	GRAY 出力画像

## リソース使用量

次の表に、Xczu9eg-ffvb1156-1-i-es1 FPGA で HD (1080x1920) 画像を処理するために、Vivado HLS 2019.1 ツールを使用して生成された異なる設定での RGB to GRAY のリソース使用量を示します。

表 154: RGB2GRAY 関数のリソース使用量のサマリ

動作モード	動作周波数 (MHz)	使用量の見積もり			
		BRAM_18K	DSP_48E	FF	LUT
1 ピクセル	300	0	3	439	280

## パフォーマンス見積もり

次の表に、Xczu9eg-ffvb1156-1-i-es1 で HD (1080x1920) 画像を処理するために、Vivado HLS 2019.1 ツールで生成された異なる設定での RGB to GRAY のパフォーマンス見積もりを示します。

表 155: RGB2GRAY 関数のパフォーマンス見積もりのサマリ

動作モード	レイテンシ見積もり
	最大レイテンシ (ms)
1 ピクセル動作 (300 MHz)	6.9

## BGR to GRAY (bgr2gray)

bgr2gray は、3 チャンネルの BGR 画像を GRAY フォーマットに変換します。

$$Y = 0.299 * R + 0.587 * G + 0.114 * B$$

説明:

- Y: グレー ピクセル
- R: 赤チャンネル
- G: 緑チャンネル
- B: 青チャンネル

## API 構文

```
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)
```

## パラメーターの説明

次の表に、テンプレートと関数のパラメーターを説明します。

表 156: bgr2gray 関数のパラメーターの説明

パラメーター	説明
SRC_T	入力ピクセルのデータ型。8ビット、符号なし、3チャンネル(XF_8UC3)のみサポート。
DST_T	出力ピクセルのデータ型。8ビット、符号なし、1チャンネル(XF_8UC1)のみサポート。
ROWS	入力および出力画像の最大高さ。8の倍数で指定。
COLS	入力および出力画像の最大幅。8の倍数で指定。
NPC	1サイクルごとに処理されるピクセル数。
_src	BGR 入力画像
_dst	GRAY 出力画像

## リソース使用量

次の表に、Xczu9eg-ffvb1156-1-i-es1 FPGA で HD (1080x1920) 画像を処理するために、Vivado HLS 2019.1 ツールを使用して生成された異なる設定での BGR to GRAY のリソース使用量を示します。

表 157: bgr2gray 関数のリソース使用量のサマリ

動作モード	動作周波数 (MHz)	使用量の見積もり			
		BRAM_18K	DSP_48E	FF	LUT
1 ピクセル	300	0	3	439	280

## パフォーマンス見積もり

次の表に、Xczu9eg-ffvb1156-1-i-es1 でグレースケール HD (1080x1920) 画像を処理するために、Vivado HLS 2019.1 ツールで生成された異なる設定での BGR to GRAY のパフォーマンス見積もりを示します。

表 158: bgr2gray 関数のパフォーマンス見積もりのサマリ

動作モード	レイテンシ見積もり
	最大レイテンシ (ms)
1 ピクセル動作 (300 MHz)	6.9

## GRAY to RGB (gray2rgb)

gray2rgb は、グレイ強度画像を RGB 色フォーマットに変換します。

$R <- Y$ 、 $G <- Y$ 、 $B <- Y$

- Y: グレー ピクセル
- R: 赤チャンネル
- G: 緑チャンネル
- B: 青チャンネル

### API 構文

```
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)
```

### パラメーターの説明

次の表に、テンプレートと関数のパラメーターを説明します。

表 159: gray2rgb 関数のパラメーターの説明

パラメーター	説明
SRC_T	入力ピクセルのデータ型。8 ビット、符号なし、1 チャンネル (XF_8UC1) のみサポート。
DST_T	出力ピクセルのデータ型。8 ビット、符号なし、3 チャンネル (XF_8UC3) のみサポート。
ROWS	入力および出力画像の最大高さ。8 の倍数で指定。
COLS	入力および出力画像の最大幅。8 の倍数で指定。
NPC	1 サイクルごとに処理されるピクセル数。
_src	グレイ入力画像。
_dst	RGB 出力画像。

### リソース使用量

次の表に、Xczu9eg-ffvb1156-1-i-es1 FPGA で HD (1080x1920) 画像を処理するために、Vivado HLS 2019.1 ツールを使用して生成された異なる設定での gray2rgb のリソース使用量を示します。

表 160: gray2rgb 関数のリソース使用量のサマリ

動作モード	動作周波数 (MHz)	使用量の見積もり			
		BRAM_18K	DSP_48E	FF	LUT
1 ピクセル	300	0	0	156	184

### パフォーマンス見積もり

次の表に、Xczu9eg-ffvb1156-1-i-es1 でグレースケール HD (1080x1920) 画像を処理するために、Vivado HLS 2019.1 ツールで生成された異なる設定での gray2rgb のパフォーマンス見積もりを示します。

表 161: gray2rgb 関数のパフォーマンス見積もりのサマリ

動作モード	レイテンシ見積もり
	最大レイテンシ (ms)
1 ピクセル動作 (300 MHz)	6.9

## GRAY to BGR (gray2bgr)

gray2bgr は、グレイ強度画像を RGB 色フォーマットに変換します。

R<-Y、G<-Y、B<-Y

説明:

- Y: グレー ピクセル
- R: 赤チャンネル
- G: 緑チャンネル
- B: 青チャンネル

### API 構文

```
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)
```

### パラメーターの説明

次の表に、テンプレートと関数のパラメーターを説明します。

表 162: gray2bgr 関数のパラメーターの説明

パラメーター	説明
SRC_T	入力ピクセルのデータ型。8 ビット、符号なし、1 チャンネル (XF_8UC1) のみサポート。
DST_T	出力ピクセルのデータ型。8 ビット、符号なし、3 チャンネル (XF_8UC3) のみサポート。
ROWS	入力および出力画像の最大高さ。8 の倍数で指定。
COLS	入力および出力画像の最大幅。8 の倍数で指定。



表 162: gray2bgr 関数のパラメーターの説明 (続き)

パラメーター	説明
NPC	1 サイクルごとに処理されるピクセル数。
_src	グレイ入力画像。
_dst	BGR 出力画像。

### リソース使用量

次の表に、Xczu9eg-ffvb1156-1-i-es1 FPGA で HD (1080x1920) 画像を処理するために、Vivado HLS 2019.1 ツールを使用して生成された異なる設定での gray2bgr のリソース使用量を示します。

表 163: gray2bgr 関数のリソース使用量のサマリ

動作モード	動作周波数 (MHz)	使用量の見積もり			
		BRAM_18K	DSP_48E	FF	LUT
1 ピクセル	300	0	0	156	184

### パフォーマンス見積もり

次の表に、Xczu9eg-ffvb1156-1-i-es1 で HD (1080x1920) 画像を処理するために、Vivado HLS 2019.1 ツールで生成された異なる設定での gray2bgr のパフォーマンス見積もりを示します。

表 164: gray2bgr 関数のパフォーマンス見積もりのサマリ

動作モード	レイテンシ見積もり
	最大レイテンシ (ms)
1 ピクセル動作 (300 MHz)	6.9

## HLS to RGB/BGR (hls2rgb、hls2bgr)

hls2 (rgb/bgr) 関数は、HLS 色空間を 3 チャンネル RGB/BGR 画像に変換します。

$$C = (1 - |2L - 1|)X S_{HSL}$$

$$H' = \frac{H}{60} \circ$$

$$X = C X (1 - |H' \bmod 2 - 1|)$$

$$(R_1, G_1, B_1) = \begin{cases} (0, 0, 0) & \text{if } H \text{ is undefined} \\ (C, X, 0) & \text{if } 0 \leq H' \leq 1 \\ (X, C, 0) & \text{if } 1 \leq H' \leq 2 \\ (0, C, X) & \text{if } 2 \leq H' \leq 3 \\ (0, X, C) & \text{if } 3 \leq H' \leq 4 \\ (X, 0, C) & \text{if } 4 \leq H' \leq 5 \\ (C, 0, X) & \text{if } 5 \leq H' \leq 6 \end{cases}$$

$$m = L - \frac{C}{2}$$

$$(R, G, B) = (R_1 + m, G_1 + m, B_1 + m)$$

## API 構文

```
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)
```

## パラメーターの説明

次の表に、テンプレートと関数のパラメーターを説明します。

表 165: HLS2RGB/BGR 関数のパラメーターの説明

パラメーター	説明
SRC_T	入力ピクセルのデータ型。8 ビット、符号なし、3 チャンネル (XF_8UC3) のみサポート。
DST_T	出力ピクセルのデータ型。8 ビット、符号なし、3 チャンネル (XF_8UC3) のみサポート。
ROWS	入力および出力画像の最大高さ。8 の倍数で指定。
COLS	入力および出力画像の最大幅。8 の倍数で指定。
NPC	1 サイクルごとに処理されるピクセル数。
_src	HLS 入力画像。
_dst	RGB/BGR 出力画像。

## リソース使用量

次の表に、Xczu9eg-ffvb1156-1-i-es1 FPGA で HD (1080x1920) 画像を処理するために、Vivado HLS 2019.1 ツールを使用して生成された異なる設定での HLS2RGB/BGRR のリソース使用量を示します。

表 166: HLS2RGB/BGR 関数のリソース使用量のサマリ

動作モード	動作周波数 (MHz)	使用量の見積もり			
		BRAM_18K	DSP_48E	FF	LUT
1 ピクセル	300	0	3	4366	3096

## パフォーマンス見積もり

次の表に、Xczu9eg-ffvb1156-1-i-es1 で HD (1080x1920) 画像を処理するために、Vivado HLS 2019.1 ツールで生成された異なる設定での HLS2RGB/BGR のパフォーマンス見積もりを示します。

表 167: HLS2RGB/BGR 関数のパフォーマンス見積もりのサマリ

動作モード	レイテンシ見積もり
	最大レイテンシ (ms)
1 ピクセル動作 (300 MHz)	6.9

## RGB to XYZ (rgb2xyz)

rgb2xyz 関数は、3 チャンネルの RGB 画像を XYZ 色空間に変換します。

$$\begin{bmatrix} X \\ Y \\ Z \end{bmatrix} = \begin{bmatrix} 0.412453 & 0.357580 & 0.180423 \\ 0.212671 & 0.715160 & 0.072169 \\ 0.019334 & 0.119193 & 0.950227 \end{bmatrix} \cdot \begin{bmatrix} R \\ G \\ B \end{bmatrix}$$

- R: 赤チャンネル
- G: 緑チャンネル
- B: 青チャンネル

### API 構文

```
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)
```

### パラメーターの説明

次の表に、テンプレートと関数のパラメーターを説明します。

表 168: RGB2XYZ 関数のパラメーターの説明

パラメーター	説明
SRC_T	入力ピクセルのデータ型。8 ビット、符号なし、3 チャンネル (XF_8UC3) のみサポート。
DST_T	出力ピクセルのデータ型。8 ビット、符号なし、3 チャンネル (XF_8UC3) のみサポート。
ROWS	入力および出力画像の最大高さ。8 の倍数で指定。
COLS	入力および出力画像の最大幅。8 の倍数で指定。
NPC	1 サイクルごとに処理されるピクセル数。
_src	RGB 入力画像。
_dst	XYZ 出力画像。

### リソース使用量

次の表に、Xczu9eg-ffvb1156-1-i-es1 FPGA で HD (1080x1920) 画像を処理するために、Vivado HLS 2019.1 ツールを使用して生成された異なる設定での RGB to XYZ のリソース使用量を示します。

表 169: RGB2XYZ 関数のリソース使用量のサマリ

動作モード	動作周波数 (MHz)	使用量の見積もり			
		BRAM_18K	DSP_48E	FF	LUT
1 ピクセル	300	0	8	644	380

### パフォーマンス見積もり

次の表に、Xczu9eg-ffvb1156-1-i-es1 で HD (1080x1920) 画像を処理するために、Vivado HLS 2019.1 ツールで生成された異なる設定での RGB to XYZ のパフォーマンス見積もりを示します。

表 170: RGB2XYZ 関数のパフォーマンス見積もりのサマリ

動作モード	レイテンシ見積もり
	最大レイテンシ (ms)
1 ピクセル動作 (300 MHz)	6.9

## BGR to XYZ (bgr2xyz)

bgr2xyz 関数は、3 チャンネルの RGB 画像を XYZ 色空間に変換します。

$$\begin{bmatrix} X \\ Y \\ Z \end{bmatrix} = \begin{bmatrix} 0.412453 & 0.357580 & 0.180423 \\ 0.212671 & 0.715160 & 0.072169 \\ 0.019334 & 0.119193 & 0.950227 \end{bmatrix} \cdot \begin{bmatrix} B \\ G \\ R \end{bmatrix}$$

- R: 赤チャンネル
- G: 緑チャンネル
- B: 青チャンネル

### API 構文

```
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)
```

### パラメーターの説明

次の表に、テンプレートと関数のパラメーターを説明します。

表 171: RGB2XYZ 関数のパラメーターの説明

パラメーター	説明
SRC_T	入力ピクセルのデータ型。8 ビット、符号なし、3 チャンネル (XF_8UC3) のみサポート。
DST_T	出力ピクセルのデータ型。8 ビット、符号なし、3 チャンネル (XF_8UC3) のみサポート。
ROWS	入力および出力画像の最大高さ。8 の倍数で指定。
COLS	入力および出力画像の最大幅。8 の倍数で指定。
NPC	1 サイクルごとに処理されるピクセル数。
_src	BGR 入力画像。
_dst	XYZ 出力画像。

### リソース使用量

次の表に、Xczu9eg-ffvb1156-1-i-es1 FPGA で HD (1080x1920) 画像を処理するために、Vivado HLS 2019.1 ツールを使用して生成された異なる設定での BGR to XYZ のリソース使用量を示します。

表 172: BGR2XYZ 関数のリソース使用量のサマリ

動作モード	動作周波数 (MHz)	使用量の見積もり			
		BRAM_18K	DSP_48E	FF	LUT
1 ピクセル	300	0	8	644	380

### パフォーマンス見積もり

次の表に、Xczu9eg-ffvb1156-1-i-es1 で HD (1080x1920) 画像を処理するために、Vivado HLS 2019.1 ツールで生成された異なる設定での BGR to XYZ のパフォーマンス見積もりを示します。

表 173: BGR2XYZ 関数のパフォーマンス見積もりのサマリ

動作モード	レイテンシ見積もり
	最大レイテンシ (ms)
1 ピクセル動作 (300 MHz)	6.9

## RGB/BGR to YCrCb (rgb2ycrcb、bgr2ycrcb)

(rgb/bgr) 2ycrcb 関数は、3 チャンネルの RGB 画像を YCrCb 色空間に変換します。

- $Y = 0.299 \cdot R + 0.587 \cdot G + 0.114 \cdot B$
- $Cr = (R - Y) \cdot 0.713 + \text{delta}$
- $Cb = (B - Y) \cdot 0.564 + \text{delta}$

$$\text{delta} = \begin{cases} 128 & \text{for 8-bit images} \\ 32768 & \text{for 16-bit images} \\ 0.5 & \text{for floating point images} \end{cases}$$

### API 構文

```
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)
```

### パラメーターの説明

次の表に、テンプレートと関数のパラメーターを説明します。

表 174: RGB/BGR2YCrCb 関数のパラメーターの説明

パラメーター	説明
SRC_T	入力ピクセルのデータ型。8 ビット、符号なし、3 チャンネル (XF_8UC3) のみサポート。

表 174: RGB/BGR2YCrCb 関数のパラメーターの説明 (続き)

パラメーター	説明
DST_T	出力ピクセルのデータ型。8 ビット、符号なし、3 チャンネル (XF_8UC3) のみサポート。
ROWS	入力および出力画像の最大高さ。8 の倍数で指定。
COLS	入力および出力画像の最大幅。8 の倍数で指定。
NPC	1 サイクルごとに処理されるピクセル数
_src	RGB/RGB 入力画像
_dst	YCrCb 出力画像

### リソース使用量

次の表に、Xczu9eg-ffvb1156-1-i-es1 FPGA で HD (1080x1920) 画像を処理するために、Vivado HLS 2019.1 ツールを使用して生成された異なる設定での RGB/BGR2YCrCb のリソース使用量を示します。

表 175: RGB/BGR2YCrCb 関数のリソース使用量のサマリ

動作モード	動作周波数 (MHz)	使用量の見積もり			
		BRAM_18K	DSP_48E	FF	LUT
1 ピクセル	300	0	5	660	500

### パフォーマンス見積もり

次の表に、Xczu9eg-ffvb1156-1-i-es1 で HD (1080x1920) 画像を処理するために、Vivado HLS 2019.1 ツールで生成された異なる設定での RGB/BGR2YCrCb のパフォーマンス見積もりを示します。

表 176: RGB/BGR2YCrCb 関数のパフォーマンス見積もりのサマリ

動作モード	レイテンシ見積もり
	最大レイテンシ (ms)
1 ピクセル動作 (300 MHz)	6.9

## RGB/BGR to HSV (rgb2hsv、bgr2hsv)

(rgb/bgr) 2hsv 関数は、3 チャンネルの RGB 画像を HSV 色空間に変換します。

$$\begin{aligned}
 V &= \max(R, G, B) \\
 S &= \begin{cases} \frac{V - \min(R, G, B)}{V} & \text{if } V \neq 0 \\ 0 & \text{otherwise} \end{cases} \\
 H &= \begin{cases} 60(G - B) / (V - \min(R, G, B)) & \text{if } V = R \\ 120 + 60(B - R) / (V - \min(R, G, B)) & \text{if } V = G \\ 240 + 60(R - G) / (V - \min(R, G, B)) & \text{if } V = B \end{cases} \\
 \text{delta} &= \begin{cases} 128 & \text{for 8-bit images} \\ 32768 & \text{for 16-bit images} \\ 0.5 & \text{for floating point images} \end{cases}
 \end{aligned}$$

## API 構文

```
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)
```

## パラメーターの説明

次の表に、テンプレートと関数のパラメーターを説明します。

表 177: RGB/BGR2HSV 関数のパラメーターの説明

パラメーター	説明
SRC_T	入力ピクセルのデータ型。8 ビット、符号なし、3 チャンネル (XF_8UC3) のみサポート。
DST_T	出力ピクセルのデータ型。8 ビット、符号なし、3 チャンネル (XF_8UC3) のみサポート。
ROWS	入力および出力画像の最大高さ。8 の倍数で指定。
COLS	入力および出力画像の最大幅。8 の倍数で指定。
NPC	1 サイクルごとに処理されるピクセル数
_src	RGB/RGB 入力画像
_dst	HSV 出力画像

## リソース使用量

次の表に、Xczu9eg-ffvb1156-1-i-es1 FPGA で HD (1080x1920) 画像を処理するために、Vivado HLS 2019.1 ツールを使用して生成された異なる設定での RGB/BGR2HSV のリソース使用量を示します。

表 178: RGB/BGR2HSV 関数のリソース使用量のサマリ

動作モード	動作周波数 (MHz)	使用量の見積もり			
		BRAM_18K	DSP_48E	FF	LUT
1 ピクセル	300	6	8	1582	1274

## パフォーマンス見積もり

次の表に、Xczu9eg-ffvb1156-1-i-es1 で HD (1080x1920) 画像を処理するために、Vivado HLS 2019.1 ツールで生成された異なる設定での RGB/BGR2HSV のパフォーマンス見積もりを示します。

表 179: RGB/BGR2HSV 関数のパフォーマンス見積もりのサマリ

動作モード	レイテンシ見積もり
	最大レイテンシ (ms)
1 ピクセル動作 (300 MHz)	6.9

## RGB/BGR to HLS (rgb2hls、bgr2hls)

(rgb/bgr)2hls 関数は、3 チャンネルの RGB 画像を HLS 色空間に変換します。

$$\begin{aligned}
 V_{max} &= \max(R, G, B) \\
 V_{min} &= \min(R, G, B) \\
 L &= \frac{V_{max} + V_{min}}{2} \\
 S &= \begin{cases} \frac{V_{max} - V_{min}}{V_{max} + V_{min}} & \text{if } L < 0.5 \\ \frac{V_{max} - V_{min}}{2 - (V_{max} + V_{min})} & \text{if } L \geq 0.5 \end{cases} \\
 H &= \begin{cases} \frac{60(G - B)}{S} & \text{if } V_{max} = R \\ 120 + \frac{60(B - R)}{S} & \text{if } V_{max} = G \\ 240 + \frac{60(R - G)}{S} & \text{if } V_{max} = B \end{cases}
 \end{aligned}$$

### API 構文

```
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)
```

### パラメーターの説明

次の表に、テンプレートと関数のパラメーターを説明します。

表 180: RGB/BGR2HLS 関数のパラメーターの説明

パラメーター	説明
SRC_T	入力ピクセルのデータ型。8 ビット、符号なし、3 チャンネル (XF_8UC3) のみサポート。
DST_T	出力ピクセルのデータ型。8 ビット、符号なし、3 チャンネル (XF_8UC3) のみサポート。
ROWS	入力および出力画像の最大高さ。8 の倍数で指定。
COLS	入力および出力画像の最大幅。8 の倍数で指定。
NPC	1 サイクルごとに処理されるピクセル数。
_src	RGB/RGB 入力画像。
_dst	HLS 出力画像。

### リソース使用量

次の表に、Xczu9eg-ffvb1156-1-i-es1 FPGA で HD (1080x1920) 画像を処理するために、Vivado HLS 2019.1 ツールを使用して生成された異なる設定での RGB/BGR2HLS のリソース使用量を示します。



表 181: RGB/BGR2HLS 関数のリソース使用量のサマリ

動作モード	動作周波数 (MHz)	使用量の見積もり			
		BRAM_18K	DSP_48E	FF	LUT
1 ピクセル	300	0	3	4366	3096

### パフォーマンス見積もり

次の表に、Xczu9eg-ffvb1156-1-i-es1 で HD (1080x1920) 画像を処理するために、Vivado HLS 2019.1 ツールで生成された異なる設定での RGB/BGR2HLS のパフォーマンス見積もりを示します。

表 182: RGB/BGR2HLS 関数のパフォーマンス見積もりのサマリ

動作モード	レイテンシ見積もり
	最大レイテンシ (ms)
1 ピクセル動作 (300 MHz)	6.9

## YCrCb to RGB/BGR (ycrcb2rgb、ycrcb2bgr)

ycrcb2 (rgb/bgr) 関数は、YCrCb 色空間を 3 チャンネル RGB/BGR 画像に変換します。

説明:

- $R = Y + 1.403 * (Cr - delta)$
- $G = Y - 0.714 * (Cr - delta) - 0.344 * (Cb - delta)$
- $B = Y + 1.773 * (Cb - delta)$

$$delta = \begin{cases} 128 & \text{for 8-bit images} \\ 32768 & \text{for 16-bit images} \\ 0.5 & \text{for floating point images} \end{cases}$$

### API 構文

```
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)
```

### パラメーターの説明

次の表に、テンプレートと関数のパラメーターを説明します。

表 183: YCrCb2RGB/BGR 関数のパラメーターの説明

パラメーター	説明
SRC_T	入力ピクセルのデータ型。8 ビット、符号なし、3 チャンネル (XF_8UC3) のみサポート。
DST_T	出力ピクセルのデータ型。8 ビット、符号なし、3 チャンネル (XF_8UC3) のみサポート。
ROWS	入力および出力画像の最大高さ。8 の倍数で指定。
COLS	入力および出力画像の最大幅。8 の倍数で指定。
NPC	1 サイクルごとに処理されるピクセル数。
_src	YCrCb 入力画像。
_dst	RGB/BGR 出力画像。

### リソース使用量

次の表に、Xczu9eg-ffvb1156-1-i-es1 FPGA で HD (1080x1920) 画像を処理するために、Vivado HLS 2019.1 ツールを使用して生成された異なる設定での YCrCb2RGB/BGR のリソース使用量を示します。

表 184: YCrCb2RGB/BGR 関数のリソース使用量のサマリ

動作モード	動作周波数 (MHz)	使用量の見積もり			
		BRAM_18K	DSP_48E	FF	LUT
1 ピクセル	300	0	4	538	575

### パフォーマンス見積もり

次の表に、Xczu9eg-ffvb1156-1-i-es1 で HD (1080x1920) 画像を処理するために、Vivado HLS 2019.1 ツールで生成された異なる設定での YCrCb2RGB/BGR のパフォーマンス見積もりを示します。

表 185: YCrCb2RGB/BGR 関数のパフォーマンス見積もりのサマリ

動作モード	レイテンシ見積もり
	最大レイテンシ (ms)
1 ピクセル動作 (300 MHz)	6.9

## HSV to RGB/BGR (hsv2rgb、hsv2bgr)

hsv2 (rgb/bgr) 関数は、HSV 色空間を 3 チャンネル RGB/BGR 画像に変換します。

$$C = V \times S_{HSV}$$

$$H' = \frac{H}{60}^{\circ}$$

$$X = C \times (1 - |H' \bmod 2 - 1|)$$

$$(R_1, G_1, B_1) = \begin{cases} (0, 0, 0) & \text{if } H \text{ is undefined} \\ (C, X, 0) & \text{if } 0 \leq H' \leq 1 \\ (X, C, 0) & \text{if } 1 \leq H' \leq 2 \\ (0, C, X) & \text{if } 2 \leq H' \leq 3 \\ (0, X, C) & \text{if } 3 \leq H' \leq 4 \\ (X, 0, C) & \text{if } 4 \leq H' \leq 5 \\ (C, 0, X) & \text{if } 5 \leq H' \leq 6 \end{cases}$$

$$m = V - C$$

$$(R, G, B) = (R_1 + m, G_1 + m, B_1 + m)$$

## API 構文

```
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)
```

## パラメーターの説明

次の表に、テンプレートと関数のパラメーターを説明します。

表 186: HSV2RGB/BGR 関数のパラメーターの説明

パラメーター	説明
SRC_T	入力ピクセルのデータ型。8ビット、符号なし、3チャンネル(XF_8UC3)のみサポート。
DST_T	出力ピクセルのデータ型。8ビット、符号なし、3チャンネル(XF_8UC3)のみサポート。
ROWS	入力および出力画像の最大高さ。8の倍数で指定。
COLS	入力および出力画像の最大幅。8の倍数で指定。
NPC	1 サイクルごとに処理されるピクセル数
_src	HSV 入力画像
_dst	RGB/BGR 出力画像

## リソース使用量

次の表に、Xczu9eg-ffvb1156-1-i-es1 FPGA で HD (1080x1920) 画像を処理するために、Vivado HLS 2019.1 ツールを使用して生成された異なる設定での HSV2RGB/BGRR のリソース使用量を示します。

表 187: HSV2RGB/BGR 関数のリソース使用量のサマリ

動作モード	動作周波数 (MHz)	使用量の見積もり			
		BRAM_18K	DSP_48E	FF	LUT
1 ピクセル	300	0	8	1543	1006

## パフォーマンス見積もり

次の表に、Xczu9eg-ffvb1156-1-i-es1 で HD (1080x1920) 画像を処理するために、Vivado HLS 2019.1 ツールで生成された異なる設定での HSV2RGB/BGR のパフォーマンス見積もりを示します。

表 188: HSV2RGB/BGR 関数のパフォーマンス見積もりのサマリ

動作モード	レイテンシ見積もり
	最大レイテンシ (ms)
1 ピクセル動作 (300 MHz)	6.9

## NV12/NV21 to RGB/BGR (nv122rgb、nv122bgr、nv212rgb、nv212bgr)

nv122rgb/nv122bgr/nv212rgb/nv212bgr は、NV12 画像フォーマットを 3 チャンネル RGB/BGR 画像に変換します。入力 は Y および UV プレーンです。NV12 にはサブサンプリングされたデータが格納されており、Y プレーンがユニット レートでサンプリングされ、2x2 の Y 値ごとに U 値と V 値が 1 つずつサンプリングされます。RGB データを生成するため、各 U および V 値が 2x2 回複製されます。

### API 構文

#### 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)
```

### パラメーターの説明

次の表に、テンプレートと関数のパラメーターを説明します。

表 189: 関数のパラメーターの説明

パラメーター	説明
SRC_T	入力ピクセルのデータ型。8 ビット、符号なし、1 チャンネル (XF_8UC1) のみサポート。
UV_T	入力ピクセルのデータ型。8 ビット、符号なし、2 チャンネル (XF_8UC2) のみサポート。
ROWS	入力および出力画像の最大高さ
COLS	入力および出力画像の最大幅。N ピクセル モードの場合は NPC の倍数で指定。
NPC	1 サイクルごとに処理される Y ピクセル数。有効なオプションは XF_NPPC1、XF_NPPC2、XF_NPPC4、および XF_NPPC8。
NPC_UV	1 サイクルごとに処理される UV ピクセル数。有効なオプションは XF_NPPC1、XF_NPPC2、および XF_NPPC4。
src_y	サイズ (ROWS, COLS) の Y 入力画像。
src_uv	サイズ (ROWS/2, COLS/2) の UV 出力画像。
_dst0	サイズ (ROWS, COLS) の出力 UV 画像。

## リソース使用量

次の表に、ザイリンクス xczu9eg-ffvb1156-2-i-es2 FPGA で HD (1080x1920) 画像を処理するために、Vivado HLS 2019.1 ツールを使用して生成された通常モード (1 ピクセル) の NV12/NV21 to RGB/ BGR 関数のリソース使用量を示します。

動作モード	動作周波数 (MHz)	使用量の見積もり				
		BRAM_18K	DSP_48E	FF	LUT	CLB
1 ピクセル	300	2	5	339	289	76

## パフォーマンス見積もり

次の表に、ザイリンクス xczu9eg-ffvb1156-2-i-es2 FPGA で HD (1080x1920) 画像を処理するために、Vivado HLS 2018.3 ツールを使用して生成された 1 ピクセル設定でのカーネルのパフォーマンス見積もりを示します。

表 190: パフォーマンス見積もりのサマリ

動作モード	レイテンシ見積もり
	最大レイテンシ (ms)
1 ピクセル動作 (300 MHz)	6.9

## NV12 to NV21、NV21 to NV12 (nv122nv21、nv212nv12)

nv122nv21/nv212nv12 関数は、NV12 (YUV4:2:0) を NV21 (YUV4:2:0) に変換、またはその逆変換を実行します。8 ビット Y プレーンの後に 2x2 サブサンプリングされたインターリーブ U/V プレーンが続きます。

## API 構文

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)
```

## パラメーターの説明

次の表に、テンプレートと関数のパラメーターを説明します。

表 191: 関数のパラメーターの説明

パラメーター	説明
SRC_Y	入力 Y ピクセルのデータ型。8 ビット、符号なし、1 チャンネル (XF_8UC1) のみサポート。
SRC_UV	入力 UV ピクセルのデータ型。8 ビット、符号なし、2 チャンネル (XF_8UC2) のみサポート。
ROWS	入力および出力画像の最大高さ
COLS	入力および出力画像の最大幅。N の倍数で指定。
NPC_Y	1 サイクルごとに処理される Y ピクセル数。有効なオプションは XF_NPPC1、XF_NPPC2、XF_NPPC4、および XF_NPPC8。
NPC_UV	1 サイクルごとに処理される UV ピクセル数。有効なオプションは XF_NPPC1、XF_NPPC2、および XF_NPPC4。
_y	Y 入力画像
_uv	UV 入力画像
out_y	Y 出力画像
out_uv	UV 出力画像

## リソース使用量

次の表に、サイリンクス xczu9eg-ffvb1156-2-i-es2 FPGA で HD (1080x1920) 画像を処理するために、Vivado HLS 2019.1 ツールを使用して生成された通常モード (1 ピクセル) の NV122NV21/NV212NV12 関数のリソース使用量を示します。

動作モード	動作周波数 (MHz)	使用量の見積もり				
		BRAM_18K	DSP_48E	FF	LUT	CLB
1 ピクセル	300	0	0	258	161	61

## パフォーマンス見積もり

次の表に、サイリンクス xczu9eg-ffvb1156-2-i-es2 FPGA で HD (1080x1920) 画像を処理するために、Vivado HLS 2019.1 ツールを使用して生成された 1 ピクセル設定でのカーネルのパフォーマンス見積もりを示します。

表 192: パフォーマンス見積もりのサマリ

動作モード	レイテンシ見積もり
	最大レイテンシ (ms)
1 ピクセル動作 (300 MHz)	6.9

## NV12/NV21 to UYVY/YUYV (nv122uyvy、nv122yuyv、nv212uyvy、nv212yuyv)

NV12/NV21 to UYVY/YUYV 関数は、NV12/NV21 (YUV4:2:0) 画像を 1 チャンネル YUYV/UYVY (YUV 4:2:2) フォーマットに変換します。YUYV はサブサンプリングフォーマットです。YUYV/UYVY は 16 ビット値、BGR は 24 ビット値です。

### API 構文

#### 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)
```

### パラメーターの説明

次の表に、テンプレートと関数のパラメーターを説明します。

表 193: 関数のパラメーターの説明

パラメーター	説明
SRC_Y	入力 Y 画像ピクセルのデータ型。8 ビット、符号なし、1 チャンネル (XF_8UC1) のみサポート。
SRC_UV	入力 UV 画像ピクセルのデータ型。8 ビット、符号なし、2 チャンネル (XF_8UC2) のみサポート。
DST_T	出力ピクセルのデータ型。16 ビット、符号なし、1 チャンネル (XF_16UC1) のみサポート。
ROWS	入力および出力画像の最大高さ。
COLS	入力および出力画像の最大幅。NPC の倍数で指定。
NPC	1 サイクルごとに処理されるピクセル数。有効なオプションは XF_NPPC1、XF_NPPC2、XF_NPPC4、および XF_NPPC8。

表 193: 関数のパラメーターの説明 (続き)

パラメーター	説明
NPC_UV	1 サイクルごとに処理されるピクセル数。有効なオプションは XF_NPPC1、XF_NPPC2、および XF_NPPC4。
_y	Y 入力画像
_uv	UV 入力画像
_dst	UYVY/YUYV 出力画像

### リソース使用量

次の表に、サイリンクス xczu9eg-ffvb1156-2-i-es2 FPGA で HD (1080x1920) 画像を処理するために、Vivado HLS 2019.1 ツールを使用して生成された通常モード (1 ピクセル) の NV12/NV21 to UYVY/YUYV 関数のリソース使用量を示します。

動作モード	動作周波数 (MHz)	使用量の見積もり				
		BRAM_18K	DSP_48E	FF	LUT	CLB
1 ピクセル	300	1	0	337	201	64

### パフォーマンス見積もり

次の表に、サイリンクス xczu9eg-ffvb1156-2-i-es2 FPGA で HD (1080x1920) 画像を処理するために、Vivado HLS 2019.1 ツールを使用して生成された 1 ピクセル設定でのカーネルのパフォーマンス見積もりを示します。

表 194: パフォーマンス見積もりのサマリ

動作モード	レイテンシ見積もり
	最大レイテンシ (ms)
1 ピクセル動作 (300 MHz)	6.9

## UYVY/YUYV to RGB/BGR (yuyv2rgb、yuyv2bgr、uyvy2rgb、uyvy2bgr)

yuyv2rgb、yuyv2bgr、uyvy2rgb、uyvy2bgr 関数は、1 チャンルの YUYV/UYVY (YUV 4:2:2) 画像フォーマットを 3 チャンルの RGB/BGR 画像に変換します。YUYV/UYVY はサブサンプリングフォーマットで、YUYV/UYVY 値のセットにより 2 つの RGB 値が得られます。YUYV/UYVY は 16 ビット値、RGB/BGR は 24 ビット値です。

### API 構文

#### 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)
```

### パラメーターの説明

次の表に、テンプレートと関数のパラメーターを説明します。

表 195: 関数のパラメーターの説明

パラメーター	説明
SRC_T	入力ピクセルのデータ型。16 ビット、符号なし、1 チャンネル (XF_16UC1) のみサポート。
DST_T	出力ピクセルのデータ型。8 ビット、符号なし、3 チャンネル (XF_8UC3) のみサポート。
ROWS	入力および出力画像の最大高さ
COLS	入力および出力画像の最大幅。N ピクセル モードの場合は NPC の倍数で指定。
NPC	1 サイクルごとに処理される Y ピクセル数。有効なオプションは XF_NPPC1、XF_NPPC2、XF_NPPC4、および XF_NPPC8。
_src	サイズ (ROWS, COLS) の入力画像。
_dst	サイズ (ROWS, COLS) の出力画像。

### リソース使用量

次の表に、ザイリンクス xczu9eg-ffvb1156-2-i-es2 FPGA で HD (1080x1920) 画像を処理するために、Vivado HLS 2019.1 ツールを使用して生成された通常モード (1 ピクセル) の UYVY/YUYV to RGB/BGR 関数のリソース使用量を示します。

動作モード	動作周波数 (MHz)	使用量の見積もり				
		BRAM_18K	DSP_48E	FF	LUT	CLB
1 ピクセル	300	0	6	444	486	109

### パフォーマンス見積もり

次の表に、ザイリンクス xczu9eg-ffvb1156-2-i-es2 FPGA で HD (1080x1920) 画像を処理するために、Vivado HLS 2019.1 ツールを使用して生成された 1 ピクセル設定でのカーネルのパフォーマンス見積もりを示します。

表 196: パフォーマンス見積もりのサマリ

動作モード	レイテンシ見積もり
	最大レイテンシ (ms)
1 ピクセル動作 (300 MHz)	6.9

## UYVY to YUYV、YUYV to UYVY (yuyv2uyvy、uyvy2yuyv)

yuyv2uyvy/uyvy2yuyv 関数は、YUYV (YUV4:2:2) を UYVY (YUV4:2:2) に変換、またはその逆変換を実行します。8 ビット Y プレーンの後に 2x2 サブサンプリングされたインターリーブ U/V プレーンが続きます。

### API 構文

#### 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)
```

### パラメーターの説明

次の表に、テンプレートと関数のパラメーターを説明します。

表 197: 関数のパラメーターの説明

パラメーター	説明
SRC_T	入力 Y ピクセルのデータ型。16 ビット、符号なし、1 チャンネル (XF_16UC1) のみサポート。
ROWS	入力および出力画像の最大高さ
COLS	入力および出力画像の最大幅。N の倍数で指定。
NPC	1 サイクルごとに処理されるピクセル数。有効なオプションは XF_NPPC1、XF_NPPC2、XF_NPPC4、および XF_NPPC8。
yuyv	入力画像
uyvy	出力画像

### リソース使用量

次の表に、サイリンクス xczu9eg-ffvb1156-2-i-es2 FPGA 用に、Vivado HLS 2019.1 ツールを使用して生成された通常モード (1 ピクセル) の UYVY to YUYV/ YUYV to UYVY 関数のリソース使用量を示します。

動作モード	動作周波数 (MHz)	使用量の見積もり				
		BRAM_18K	DSP_48E	FF	LUT	CLB
1 ピクセル	300	0	1	368	176	109

## パフォーマンス見積もり

次の表に、ザイリンクス xczu9eg-ffvb1156-2-i-es2 FPGA でグレースケール HD (1080x1920) 画像を処理するために、Vivado HLS 2019.1 ツールを使用して生成された 1 ピクセル設定でのカーネルのパフォーマンス見積もりを示します。

表 198: パフォーマンス見積もりのサマリ

動作モード	レイテンシ見積もり
	最大レイテンシ (ms)
1 ピクセル動作 (300 MHz)	6.9

## UYVY/YUYV to NV21 (uyvy2nv21、yuyv2nv21)

UYVY/YUYV2NV21 関数は、1 チャンネルの YUYV/UYVY (YUV 4:2:2) 画像フォーマットを NV21 (YUV 4:2:0) フォーマットに変換します。YUYV/UYVY はサブサンプリングフォーマットで、YUYV/UYVY 値の 1 セットから Y 値が 2 つと U 値および V 値が 1 つずつ得られます。

### API 構文

#### 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)
```

### パラメーターの説明

次の表に、テンプレートと関数のパラメーターを説明します。

表 199: 関数のパラメーターの説明

パラメーター	説明
SRC_T	入力ピクセルのデータ型。16 ビット、符号なし、1 チャンネル (XF_16UC1) のみサポート。
Y_T	出力 Y 画像ピクセルのデータ型。8 ビット、符号なし、1 チャンネル (XF_8UC1) のみサポート。
UV_T	出力 UV 画像ピクセルのデータ型。8 ビット、符号なし、2 チャンネル (XF_8UC2) のみサポート。
ROWS	入力および出力画像の最大高さ。
COLS	入力および出力画像の最大幅。NPC の倍数で指定。
NPC	1 サイクルごとに処理されるピクセル数。可能なオプションは XF_NPPC1、XF_NPPC2、XF_NPPC4、および XF_NPPC8。
NPC_UV	1 サイクルごとに処理される U、V ピクセル数。可能なオプションは XF_NPPC1、XF_NPPC2、および XF_NPPC4。
_src	入力画像

表 199: 関数のパラメーターの説明 (続き)

パラメーター	説明
_y_image	Y 出力画像
_uv_image	UV 出力画像

### リソース使用量

次の表に、ザイリンクス xczu9eg-ffvb1156-2-i-es2 FPGA で HD (1080x1920) 画像を処理するために、Vivado HLS 2019.1 ツールを使用して生成された通常モード (1 ピクセル) の UYVY/YUYV to NV21 関数のリソース使用量を示します。

動作モード	動作周波数 (MHz)	使用量の見積もり				
		BRAM_18K	DSP_48E	FF	LUT	CLB
1 ピクセル	300	0	0	215	73	42

### パフォーマンス見積もり

次の表に、ザイリンクス xczu9eg-ffvb1156-2-i-es2 FPGA で HD (1080x1920) 画像を処理するために、Vivado HLS 2019.1 ツールを使用して生成された 1 ピクセル設定でのカーネルのパフォーマンス見積もりを示します。

表 200: パフォーマンス見積もりのサマリ

動作モード	レイテンシ見積もり
	最大レイテンシ (ms)
1 ピクセル動作 (300 MHz)	6.9

## RGB/BGR to NV12/NV21 (rgb2nv12、bgr2nv12、rgb2nv21、bgr2nv21)

rgb2nv12/bgr2nv12/rgb2nv21/bgr2nv21 は、3 チャンネルの RGB/BGR 画像を NV12/NV21 (4:2:0) フォーマットに変換します。出力は Y プレーンおよびインターリーブされた UV/VU プレーンです。NV12/NV21 にはサブサンプリングされたデータが格納されるので、Y は RGB/BGR の 1 ピクセルごとにサンプリングされ、U および V は 2 行と 2 列 (2x2) のピクセルごとにサンプリングされます。UV/VU プレーンのサイズは (rows/2)\*(columns/2) で、U と V の値がインターリーブされます。

### API 構文

#### 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)
```

## パラメーターの説明

次の表に、テンプレートと関数のパラメーターを説明します。

表 201: 関数のパラメーターの説明

パラメーター	説明
SRC_T	入力ピクセルのデータ型。8ビット、符号なし、3チャンネル (XF_8UC3) のみサポート。
Y_T	出力ピクセルのデータ型。8ビット、符号なし、1チャンネル (XF_8UC1) のみサポート。
UV_T	出力ピクセルのデータ型。8ビット、符号なし、2チャンネル (XF_8UC2) のみサポート。
ROWS	入力および出力画像の最大高さ
COLS	入力および出力画像の最大幅。N ピクセル モードの場合は NPC の倍数で指定。
NPC	1 サイクルごとに処理されるピクセル数。有効なオプションは XF_NPPC1、XF_NPPC2、XF_NPPC4、および XF_NPPC8。
NPC_UV	1 サイクルごとに処理されるピクセル数。有効なオプションは XF_NPPC1、XF_NPPC2、および XF_NPPC4。
_src	サイズ (ROWS, COLS) の RGB 入力画像。
_y	サイズ (ROWS, COLS) の出力 Y 画像。
_uv	サイズ (ROWS/2, COLS/2) の出力 UV 画像。

## リソース使用量

次の表に、サイリンクス xczu9eg-ffvb1156-2-i-es2 FPGA で HD (1080x1920) 画像を処理するために、Vivado HLS 2019.1 ツールを使用して生成された通常モード (1 ピクセル) の RGB/BGR to NV12/NV21 関数のリソース使用量を示します。

動作モード	動作周波数 (MHz)	使用量の見積もり				
		BRAM_18K	DSP_48E	FF	LUT	CLB
1 ピクセル	300	0	9	413	279	66

## パフォーマンス見積もり

次の表に、サイリンクス xczu9eg-ffvb1156-2-i-es2 FPGA で HD (1080x1920) 画像を処理するために、Vivado HLS 2019.1 ツールを使用して生成された 1 ピクセル設定でのカーネルのパフォーマンス見積もりを示します。

表 202: パフォーマンス見積もりのサマリ

動作モード	レイテンシ見積もり
	最大レイテンシ (ms)
1 ピクセル動作 (300 MHz)	6.9

## BGR to RGB、RGB to BGR (bgr2rgb、rgb2bgr)

bgr2rgb/rgb2bgr は、3 チャンネルの BGR を RGB フォーマットに、または RGB を BGR フォーマットに変換します。

### API 構文

```
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)
```

### パラメーターの説明

次の表に、テンプレートと関数のパラメーターを説明します。

表 203: 関数のパラメーターの説明

パラメーター	説明
SRC_T	入力ピクセルのデータ型。8 ビット、符号なし、3 チャンネル (XF_8UC3) のみサポート。
DST_T	出力ピクセルのデータ型。8 ビット、符号なし、3 チャンネル (XF_8UC3) のみサポート。
ROWS	入力および出力画像の最大高さ。
COLS	入力および出力画像の最大幅。N の倍数で指定。
NPC	1 サイクルごとに処理されるピクセル数。有効なオプションは XF_NPPC1、XF_NPPC2、XF_NPPC4、および XF_NPPC8。
_src	BGR/RGB 入力画像
_dst	RGB/BGR 出力画像

### リソース使用量

次の表に、サイリンクス xczu9eg-ffvb1156-2-i-es2 FPGA 用に、Vivado HLS 2019.1 ツールを使用して生成された通常モード (1 ピクセル) の RGB to BGR/BGR to RGB 関数のリソース使用量を示します。

動作モード	動作周波数 (MHz)	使用量の見積もり				
		BRAM_18K	DSP_48E	FF	LUT	CLB
1 ピクセル	300	0	0	317	118	98

## パフォーマンス見積もり

次の表に、ザイリンクス xczu9eg-ffvb1156-2-i-es2 FPGA で HD (1080x1920) 画像を処理するために、Vivado HLS 2019.1 ツールを使用して生成された 1 ピクセル設定でのカーネルのパフォーマンス見積もりを示します。

表 204: パフォーマンス見積もりのサマリ

動作モード	レイテンシ見積もり
	最大レイテンシ (ms)
1 ピクセル動作 (300 MHz)	6.9

## RGB/BGR to UYVY/YUYV (rgb2uyvy、rgb2yuyv、bgr2uyvy、bgr2yuyv)

RGB/BGR to UYVY/YUYV 関数は、3 チャンネルの RGB/BGR 画像を 1 チャンネル YUYV/UYVY (YUV 4:2:2) フォーマットに変換します。YUYV はサブサンプリング フォーマットで、2 つの RGB 値のセットにより YUYV/UYVY 値が得られます。YUYV/UYVY は 16 ビット値、BGR は 24 ビット値です。

### API 構文

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)
```

### パラメーターの説明

次の表に、テンプレートと関数のパラメーターを説明します。

表 205: 関数のパラメーターの説明

パラメーター	説明
SRC_T	入力ピクセルのデータ型。8 ビット、符号なし、3 チャンネル (XF_8UC3) のみサポート。
DST_T	出力ピクセルのデータ型。16 ビット、符号なし、1 チャンネル (XF_16UC1) のみサポート。

表 205: 関数のパラメーターの説明 (続き)

パラメーター	説明
ROWS	入力および出力画像の最大高さ
COLS	入力および出力画像の最大幅。NPC の倍数で指定。
NPC	1 サイクルごとに処理されるピクセル数。有効なオプションは XF_NPPC1、XF_NPPC2、XF_NPPC4、および XF_NPPC8。
_src	RGB/RGB 入力画像
_dst	UYVY/YUYV 出力画像

## リソース使用量

次の表に、ザイリンクス xczu9eg-ffvb1156-2-i-es2 FPGA 用に、Vivado HLS 2019.1 ツールを使用して生成された通常モード (1 ピクセル) の RGB/BGR to UYVY/YUYV 関数のリソース使用量を示します。

動作モード	動作周波数 (MHz)	使用量の見積もり				
		BRAM_18K	DSP_48E	FF	LUT	CLB
1 ピクセル	300	0	9	249	203	55

## パフォーマンス見積もり

次の表に、ザイリンクス xczu9eg-ffvb1156-2-i-es2 FPGA で HD (1080x1920) 画像を処理するために、Vivado HLS 2019.1 ツールを使用して生成された 1 ピクセル設定でのカーネルのパフォーマンス見積もりを示します。

表 206: パフォーマンス見積もりのサマリ

動作モード	レイテンシ見積もり
	最大レイテンシ (ms)
1 ピクセル動作 (300 MHz)	6.9

## XYZ to RGB/BGR (xyz2rgb、xyz2bgr)

xyz2rgb 関数は、XYZ 色空間を 3 チャンネル RGBA 画像に変換します。

$$\begin{bmatrix} R \\ G \\ B \end{bmatrix} = \begin{bmatrix} 3.240479 & -1.53715 & -0.498535 \\ -0.969256 & 1.875991 & 0.041556 \\ 0.055648 & -0.204043 & 1.057311 \end{bmatrix} \cdot \begin{bmatrix} X \\ Y \\ Z \end{bmatrix}$$

## API 構文

```
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)
```

## パラメーターの説明

次の表に、テンプレートと関数のパラメーターを説明します。



表 207: XYZ2RGB/BGR 関数のパラメーターの説明

パラメーター	説明
SRC_T	入力ピクセルのデータ型。8ビット、符号なし、3チャンネル (XF_8UC3) のみサポート。
DST_T	出力ピクセルのデータ型。8ビット、符号なし、3チャンネル (XF_8UC3) のみサポート。
ROWS	入力および出力画像の最大高さ。8の倍数で指定。
COLS	入力および出力画像の最大幅。8の倍数で指定。
NPC	1 サイクルごとに処理されるピクセル数。
_src	XYZ 入力画像。
_dst	RGB/BGR 出力画像。

### リソース使用量

次の表に、Xczu9eg-ffvb1156-1-i-es1 FPGA で HD (1080x1920) 画像を処理するために、Vivado HLS 2019.1 ツールを使用して生成された異なる設定での XYZ2RGB/BGR のリソース使用量を示します。

表 208: XYZ2RGB/BGR 関数のリソース使用量のサマリ

動作モード	動作周波数 (MHz)	使用量の見積もり			
		BRAM_18K	DSP_48E	FF	LUT
1 ピクセル	300	0	8	639	401

### パフォーマンス見積もり

次の表に、Xczu9eg-ffvb1156-1-i-es1 で HD (1080x1920) 画像を処理するために、Vivado HLS 2019.1 ツールで生成された異なる設定での XYZ2RGB/BGR のパフォーマンス見積もりを示します。

表 209: XYZ2RGB/BGR 関数のパフォーマンス見積もりのサマリ

動作モード	レイテンシ見積もり
	最大レイテンシ (ms)
1 ピクセル動作 (300 MHz)	6.9

## 色しきい値処理 (colorthresholding)

colorthresholding 関数は、ソース画像の色空間値を低しきい値と高しきい値を使用して比較し、出力として 255 または 0 を返します。

### API 構文

```
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])
```

### パラメーターの説明

次の表に、テンプレートと関数のパラメーターを説明します。

パラメーター	説明
SRC_T	入力ピクセルのデータ型。8ビット、符号なし、3チャンネル (XF_8UC3) のみサポート。
DST_T	出力ピクセルのデータ型。8ビット、符号なし、1チャンネル (XF_8UC1) のみサポート。
MAXCOLORS	カラー値の最大数
ROWS	入力および出力画像の最大高さ
COLS	入力および出力画像の最大幅。8ピクセル モードの場合は8の倍数で指定。
NPC	1サイクルごとに処理されるピクセル数。XF_NPPC1 のみサポート。
_src_mat	入力画像
_dst_mat	しきい値画像
low_thresh	色の最小しきい値
high_thresh	色の最大しきい値

## 比較 (compare)

compare 関数は、2 のつ入力画像 src1 と src2 の各要素を比較し、結果を dst に保存します。

$\text{dst}(x,y) = \text{src1}(x,y) \text{ CMP\_OP } \text{src2}(x,y)$

CMP\_OP: ピクセル間の関係を指定するフラグ。

- XF\_CMP\_EQ: src1 と src2 は等価
- XF\_CMP\_GT: src1 は src2 より大きい
- XF\_CMP\_GE: src1 は src2 以上
- XF\_CMP\_LT: src1 は src2 より小さい
- XF\_CMP\_LE: src1 は src2 以下
- XF\_CMP\_NE: src1 と src2 は不等価

比較結果が真の場合は dst の対応する要素が 255 になり、偽の場合は 0 になります。

### API 構文

```
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)
```

### パラメーターの説明

次の表に、テンプレートと関数のパラメーターを説明します。

表 210: compare 関数のパラメーターの説明

パラメーター	説明
CMP_OP	チェックする要素間の関係を指定するフラグ

表 210: compare 関数のパラメーターの説明 (続き)

パラメーター	説明
SRC_T	入力ピクセルのデータ型。8 ビット、符号なし、1 チャンネル (XF_8UC1) をサポート。
ROWS	入力および出力画像の最大高さ。
COLS	入力および出力画像の最大幅。N ピクセルの並列処理の場合、幅は N の倍数で指定。
NPC	1 サイクルごとに処理されるピクセル数。1 ピクセルの場合は XF_NPPC1、8 ピクセルの場合は XF_NPPC8。
_src1	1 つ目の入力画像
_src2	2 つ目の入力画像
_dst	出力画像

### リソース使用量

次の表に、Xczu9eg-ffvb1156-1-i-es1 FPGA で Vivado HLS 2019.1 バージョン ツールを使用して生成された、リソース最適化 (8 ピクセル) モードおよび通常モードにおける XF\_CMP\_NE 設定の Compare 関数のリソース使用量を示します。

表 211: compare 関数のリソース使用量のサマリ

名前	リソース使用量	
	1 クロックごとに 1 ピクセル動作	8 クロックごとに 1 ピクセル動作
	300 MHz	150 MHz
BRAM_18K	0	0
DSP48E	0	0
FF	87	60
LUT	38	84
CLB	16	20

### パフォーマンス見積もり

次の表に、Xczu9eg-ffvb1156-1-i-es1 FPGA でグレースケール HD (1080x1920) 画像を処理するために、Vivado HLS 2019.1 ツールを使用して生成された異なる設定でのカーネルのパフォーマンス見積もりを示します。

表 212: compare 関数のパフォーマンス見積もりのサマリ

動作モード	レイテンシ見積もり	
	動作周波数 (MHz)	レイテンシ (ms)
1 ピクセル	300	6.9
8 ピクセル	150	1.7

## スカラー値との比較 (compareS)

compareS 関数は、入力画像 (src1) のピクセルをスカラー値 scl と比較し、結果を dst に保存します。

`dst(x,y)=src1(x,y) CMP_OP scalar`

CMP\_OP: ピクセルとスカラーの関係を指定するフラグ。

- XF\_CMP\_EQ: src1 と scl は等価
- XF\_CMP\_GT: src1 は scl より大きい
- XF\_CMP\_GE: src1 は scl 以上
- XF\_CMP\_LT: src1 は scl より小さい
- XF\_CMP\_LE: src1 は scl 以下
- XF\_CMP\_NE: src1 と scl は不等価

比較結果が真の場合は dst の対応する要素が 255 になり、偽の場合は 0 になります。

## API 構文

```
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
_dst[XF_CHANNELS(SRC_T, NPC)], xf::Mat<SRC_T, ROWS, COLS, NPC> &_dst)
```

## パラメーターの説明

次の表に、テンプレートと関数のパラメーターを説明します。

表 213: compareS 関数のパラメーターの説明

パラメーター	説明
CMP_OP	チェックする要素とスカラーの関係を指定するフラグ
SRC_T	入力ピクセルのデータ型。8 ビット、符号なし、1 チャンネル (XF_8UC1) をサポート。
ROWS	入力および出力画像の最大高さ
COLS	入力および出力画像の最大幅。N ピクセルの並列処理の場合、幅は N の倍数で指定。
NPC	1 サイクルごとに処理されるピクセル数。1 ピクセル動作の場合は XF_NPPC1、8 ピクセル動作の場合は XF_NPPC8。
_src1	1 つ目の入力画像
_scl	入力スカラー値。サイズはチャンネル数にする必要があります。
_dst	出力画像

## リソース使用量

次の表に、Xczu9eg-ffvb1156-1-i-es1 FPGA で Vivado HLS 2019.1 バージョン ツールを使用して生成された、リソース最適化 (8 ピクセル) モードおよび通常モードにおける XF\_CMP\_NE 設定の CompareS 関数のリソース使用量を示します。

表 214: compareS 関数のリソース使用量のサマリ

名前	リソース使用量	
	1 クロックごとに 1 ピクセル動作	8 クロックごとに 1 ピクセル動作
	300 MHz	150 MHz
BRAM_18K	0	0
DSP48E	0	0
FF	93	93

表 214: compareS 関数のリソース使用量のサマリ (続き)

名前	リソース使用量	
	1 クロックごとに 1 ピクセル動作	8 クロックごとに 1 ピクセル動作
	300 MHz	150 MHz
LUT	39	68
CLB	21	28

### パフォーマンス見積もり

次の表に、Xczu9eg-ffvb1156-1-i-es1 FPGA でグレースケール HD (1080x1920) 画像を処理するために、Vivado HLS 2019.1 ツールを使用して生成された異なる設定でのカーネルのパフォーマンス見積もりを示します。

表 215: compareS 関数のパフォーマンス見積もりのサマリ

動作モード	レイテンシ見積もり	
	動作周波数 (MHz)	レイテンシ (ms)
1 ピクセル	300	6.9
8 ピクセル	150	1.7

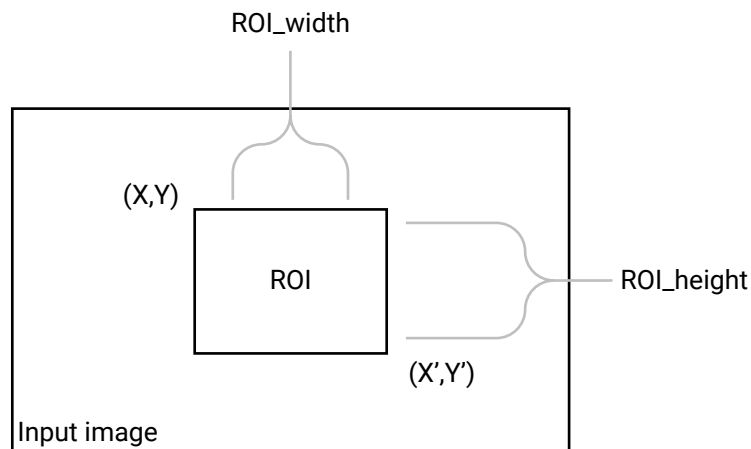
## 切り抜き (crop)

Crop 関数は、入力画像から関心領域 (ROI) を抽出します。

$$P(X,Y) \leq P(x_i, y_i) \leq P(X',Y')$$

- P(X,Y): ROI の左上角
- P(X',Y'): ROI の右下角

図 9: crop 関数



X22036-112718

## API 構文

```
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)
```

## パラメーターの説明

次の表に、テンプレートと関数のパラメーターを説明します。

表 216: crop 関数のパラメーターの説明

パラメーター	説明
SRC_T	入力ピクセルのデータ型。8 ビット、符号なし、1 および 3 チャンネル (XF_8UC1、XF_8UC3) のみサポート。
ROWS	入力および出力画像の最大高さ。
COLS	入力および出力画像の最大幅。8 ピクセル動作の場合は 8 の倍数で指定。
ARCH_TYPE	アーキテクチャタイプ。ストリーミングインプリメンテーションの場合は 0、メモリマップドインプリメンテーションの場合は 1 を指定します。
NPC	1 サイクルごとに処理されるピクセル数。2 のべき数を指定する必要があります。
_src_mat	入力画像
_dst_mat	出力 ROI 画像
roi	矩形の左上角、高さ、および幅で構成される xf::Rect オブジェクト。

## リソース使用量

次の表に、3 つの ROI (480x640、100x200、300x300) を処理するために xczu9eg-ffvb1156-2-i-es2 FPGA 用に Vivado HLS 2019.1 ツールを使用して生成された通常モードの crop 関数のリソース使用量を示します。

表 217: crop 関数のリソース使用量のサマリ

名前	リソース使用量	
	1 クロックごとに 1 ピクセル動作	8 クロックごとに 1 ピクセル動作
	300 MHz	300 MHz
BRAM_18K	6	8
DSP48E	10	10
FF	17482	16995
LUT	16831	15305

## パフォーマンス見積もり

次の表に、Xczu9eg-ffvb1156-1-i-es1 FPGA で 3 つの ROI (480x640、100x200、300x300) のグレースケール HD (1080x1920) 画像を処理するために、Vivado HLS 2019.1 ツールを使用して生成された異なる設定でのカーネルのパフォーマンス見積もりを示します。

表 218: crop 関数のパフォーマンス見積もりのサマリ

動作モード	レイテンシ見積もり	
	動作周波数 (MHz)	レイテンシ (ms)
1 ピクセル	300	1.7
8 ピクセル	300	0.6

### 複数 ROI 抽出

accel.cpp で `xf::crop` 関数を複数呼び出すことができます。

### 複数 ROI 抽出の例

```
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]);}
```

## カスタムたたみ込み (filter2D)

`filter2D` 関数は、ユーザー定義のカーネルを使用して画像に対してたたみ込みを実行します。

たたみ込みは、2つの関数 `f` および `g` に対する数学演算で、3つ目の関数を生成します。通常 3つ目の関数は元の関数の 1つを変更したバージョンと考慮され、元の関数の 1つが変換された量までの 2つの関数のエリア オーバーラップを示します。

フィルターには、ユニティ ゲイン フィルターまたは非ユニティ ゲイン フィルターを使用できます。フィルターは、XF\_16SP 型である必要があります。係数が浮動小数点の場合は、`Qm.n` に変換して入力として供給し、シフトパラメーターを `n` 値に設定する必要があります。入力が浮動小数点でない場合は、フィルターは直接供給され、シフトパラメーターは 0 に設定されます。

### API 構文

```
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)
```

### パラメーターの説明

次の表に、テンプレートと関数のパラメーターを説明します。

表 219: filter2D 関数のパラメーターの説明

パラメーター	説明
BORDER_TYPE	サポートされる境界タイプは XF_BORDER_CONSTANT。
FILTER_HEIGHT	入力フィルターの行数

表 219: filter2D 関数のパラメーターの説明 (続き)

パラメーター	説明
FILTER_WIDTH	入力フィルターの列数
SRC_T	入力ピクセルのデータ型。8 ビット、符号なし、1 および 3 チャンネル (XF_8UC1、XF_8UC3) のみサポート。
DST_T	出力ピクセルのデータ型。8 ビット、符号なし、1 チャンネルおよび 3 チャンネル (XF_8UC1、XF_8UC3) および 16 ビット、符号付き、1 チャンネルおよび 3 チャンネル (XF_16SC1、XF_16SC3) をサポート。
ROWS	入力および出力画像の最大高さ
COLS	入力および出力画像の最大幅。8 ピクセル モードの場合は 8 の倍数で指定。
NPC	1 サイクルごとに処理されるピクセル数。1 ピクセルの場合は XF_NPPC1、8 ピクセルの場合は XF_NPPC8。
_src_mat	入力画像
_dst_mat	出力画像
filter	任意のサイズ (奇数) の入力フィルター。フィルターの係数は 16 ビット値または 16 ビット固定小数点の等価値。
_shift	フィルターは、XF_16SP 型である必要があります。係数が浮動小数点の場合は、Qm.n に変換して入力として供給し、シフトパラメーターを n 値に設定する必要があります。入力が浮動小数点でない場合は、フィルターは直接供給され、シフトパラメーターは 0 に設定されます。

## リソース使用量

次の表に、Xczu9eg-ffvb1156-1-i-es1 FPGA でグレースケール HD (1080x1920) 画像を処理するために、Vivado HLS 2019.1 ツールを使用して生成された異なる設定でのカーネルのリソース使用量を示します。

表 220: filter2D 関数のリソース使用量のサマリ

動作モード	フィルター サイズ	動作周波数 (MHz)	使用量の見積もり				
			BRAM_18K	DSP_48E	FF	LUT	CLB
1 ピクセル	3x3	300	3	9	1701	1161	269
	5x5	300	5	25	3115	2144	524
8 ピクセル	3x3	150	6	72	2783	2768	638
	5x5	150	10	216	3020	4443	1007

次の表に、ザイリンクス Xczu9eg-ffvb1156-1-i-es1 FPGA で 3 チャンネルの 4K 画像を処理するために、Vivado HLS 2019.1 ツールを使用して生成された異なる設定でのカーネルのリソース使用量を示します。

表 221: filter2D 関数のリソース使用量のサマリ

動作モード	フィルター サイズ	動作周波数 (MHz)	使用量の見積もり			
			BRAM_18K	DSP_48E	FF	LUT
1 ピクセル	3x3	300	18	27	886	801
	5x5	300	30	75	1793	1445



## パフォーマンス見積もり

次の表に、Xczu9eg-ffvb1156-1-i-es1 でグレースケール HD (1080x1920) 画像を処理するために、Vivado HLS 2019.1 ツールを使用して生成された異なる設定でのカーネルのパフォーマンス見積もりを示します。

表 222: filter2D 関数のパフォーマンス見積もりのサマリ

動作モード	動作周波数 (MHz)	フィルター サイズ	レイテンシ見積もり
			最大 (ms)
1 ピクセル	300	3x3	7
	300	5x5	7.1
8 ピクセル	150	3x3	1.86
	150	5x5	1.86

## 遅延 (delayMat)

画像処理パイプラインでは、関数への入力が入力 FIFO インターフェイスを使用して同期化されていない可能性があります。最初の入力の最初のデータ パケットが、2 番目の入力の最初のデータ パケットより数クロック サイクル後に到着することがあります。関数のインターフェイスに十分な深さの FIFO がないと、ハードウェアでデザイン全体が停止します。入力を同期化するため、早く到着する入力パケットを指定したクロック サイクル数遅延させるこの関数が提供されています。

## API 構文

```
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)
```

## パラメーターの説明

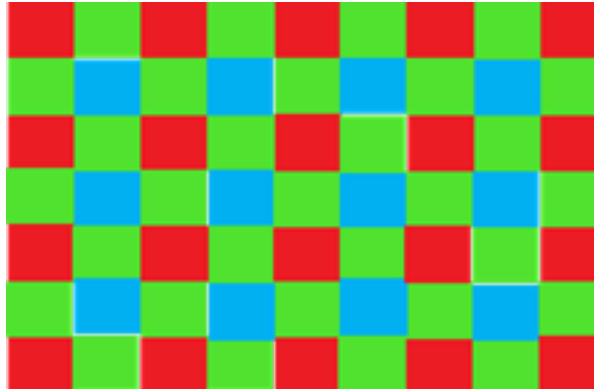
次の表に、テンプレートと関数のパラメーターを説明します。

パラメーター	説明
SRC_T	入力および出力のピクセルのデータ型
ROWS	入力および出力画像の最大高さ。
COLS	入力および出力画像の最大幅 (8 ピクセル動作の場合は 8 の倍数で指定)
NPC	1 サイクルごとに処理されるピクセル数。1 ピクセルの場合は XF_NPPC1、8 ピクセルの場合は XF_NPPC8。
MAXDELAY	関数にインスタンス化するための最大遅延。
_src	入力画像
_dst	出力画像

## モザイク処理 (demosaicing)

demosaicing 関数は、デジタル カメラ センサーからの 1 プレーン ベイヤー パターン出力をカラー画像に変換します。Malvar、He、および Cutler により提唱されたバイリニア補間手法を改善したものをインプリメントします。

図 10: カラー画像のバイヤー モザイク



上の図は、シングル CCD デジタルカメラのカラー画像キャプチャを示しています。

### API 構文

```
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)
```

### パラメーターの説明

次の表に、テンプレートと関数のパラメーターを説明します。

表 223: demosaicing 関数のパラメーターの説明

パラメーター	説明
BFORMAT	入力バイヤー パターン。サポートされる値は XF_BAYER_BG、XF_BAYER_GB、XF_BAYER_GR、および XF_BAYER_RG です。
SRC_T	入力ピクセルのデータ型。8 ビット、符号なし、1 チャネルと 3 チャネル (XF_8UC1、XF_8UC3)、および 16 ビット、符号なし、1 チャネルと 3 チャネル (XF_16UC1、XF_16UC3) をサポート。
DST_T	出力ピクセルのデータ型。8 ビット、符号なし、4 チャネル (XF_8UC4)、および 16 ビット、符号なし、4 チャネル (XF_16UC4) をサポート。
ROWS	処理される画像の行数。
COLS	処理される画像の列数。8 ピクセル モードの場合は 8 の倍数で指定。
NPC	1 サイクルごとに処理されるピクセル数。1 ピクセル並列処理 (XF_NPPC1)、2 ピクセル並列処理 (XF_NPPC2)、および 4 ピクセル並列処理 (XF_NPPC4) をサポート。ピクセルのデータ型 XF_16UC1 では XF_NPPC4 はサポートされません。
USE_URAM	ストレージ構造を UltraRAM にマップ。
_src_mat	入力画像
_dst_mat	出力画像

### リソース使用量

次の表に、Xczu9eg-ffvb1156-1-i-es1 FPGA で Vivado HLS 2019.1 バージョン ツールを使用して生成された demosaicing 関数のリソース使用量を示します。

表 224: demosaicing 関数のリソース使用量のサマリ

動作モード	動作周波数 (MHz)	使用量の見積もり				
		BRAM_18K	DSP48E	FF	LUT	CLB
1 ピクセル	300	8	0	1906	1915	412
2 ピクセル	300	8	0	2876	3209	627
4 ピクセル	300	8	0	2950	3222	660

次の表に、xczu7ev-ffvc1156-2-e FPGA で SDx 2019.1 バージョン ツールを使用して生成された demosaicing 関数のリソース使用量を示します。

表 225: UltraRAM をイネーブルにした場合の demosaicing 関数のリソース使用量のサマリ

動作モード	動作周波数 (MHz)	使用量の見積もり					
		BRAM_18K	URAM	DSP48E	FF	LUT	CLB
1 ピクセル	300	0	1	0	1366	1339	412

### パフォーマンス見積もり

次の表に、Xczu9eg-ffvb1156-1-i-es1 で 4K (3840x2160) 画像を処理するために、Vivado HLS 2019.1 ツールを使用して生成された異なる設定でのパフォーマンス見積もりを示します。

表 226: demosaicing 関数のパフォーマンス見積もりのサマリ

動作モード	レイテンシ見積もり
	最大レイテンシ (ms)
1 ピクセル動作 (300 MHz)	27.82
2 ピクセル動作 (300 MHz)	13.9
4 ピクセル動作 (300 MHz、8 ビット画像のみ)	6.95

## 膨張 (dilate)

膨張演算は、現在のピクセル強度をその近傍 nxn ピクセルで最大の強度値に置き換えます。

$$dst(x, y) = \max_{\substack{x-1 \leq x' \leq x+1 \\ y-1 \leq y' \leq y+1}} src(x', y')$$

### API 構文

```
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])
```

## パラメーターの説明

次の表に、テンプレートと関数のパラメーターを説明します。

表 227: dilate 関数のパラメーターの説明

パラメーター	説明
BORDER_TYPE	サポートされる境界タイプは XF_BORDER_CONSTANT。
TYPE	入力および出力のピクセルのデータ型。8 ビット、符号なし、1 および 3 チャンネル (XF_8UC1、XF_8UC3) のみサポート。
ROWS	入力および出力画像の最大高さ。
COLS	入力および出力画像の最大幅 (8 ピクセル動作の場合は 8 の倍数で指定)
K_SHAPE	カーネルの形状。サポートされるカーネルの形状は RECT、CROSS、および ELLIPSE です。
NPC	1 サイクルごとに処理されるピクセル数。1 ピクセルの場合は XF_NPPC1、8 ピクセルの場合は XF_NPPC8。
K_ROWS	カーネルの高さ。
K_COLS	カーネルの幅。
ITERATIONS	膨張の適用回数。現在のところ、矩形 (RECT) のカーネル要素でのみサポートされます。
_src_mat	入力画像
_dst_mat	出力画像
_kernel	サイズ K_ROWS * K_COLS の膨張カーネル。

## リソース使用量

次の表に、Xczu9eg-ffvb1156-1-i-es1 FPGA で HD 画像 (1080X1920) を処理するために、Vivado HLS 2019.1 バージョンツールで矩形 (RECT) 構造要素を使用して生成された膨張関数のリソース使用量を示します。

表 228: dilate 関数のリソース使用量のサマリ

名前	リソース使用量	
	1 クロックごとに 1 ピクセル動作	8 クロックごとに 1 ピクセル動作
	300 MHz	150 MHz
BRAM_18K	3	6
DSP48E	0	0
FF	411	657
LUT	392	1249
CLB	96	255

次の表に、Xczu9eg-ffvb1156-1-i-es1 FPGA で 3 チャンネルの 4K 画像を処理するために、Vivado HLS 2019.1 バージョンツールでの矩形 (RECT) 構造要素を使用して 1 ピクセル操作で生成された膨張関数のリソース使用量を示します。

表 229: dilate 関数のリソース使用量のサマリ

名前	リソース使用量	
	1 クロックごとに 1 ピクセル動作	
	300 MHz	
BRAM_18K	18	

表 229: dilate 関数のリソース使用量のサマリ (続き)

名前	リソース使用量
	1 クロックごとに 1 ピクセル動作
	300 MHz
DSP48E	0
FF	983
LUT	745
CLB	186

### パフォーマンス見積もり

次の表に、Xczu9eg-ffvb1156-1-i-es1 FPGA で Vivado HLS 2019.1 ツールを使用して生成された、通常動作 (1 ピクセル) およびリソース最適化 (8 ピクセル) 設定での膨張関数のパフォーマンス見積もりを示します。

表 230: dilate 関数のパフォーマンス見積もりのサマリ

動作モード	レイテンシ見積もり	
	最小 (ms)	最大 (ms)
1 ピクセル (300 MHz)	7.0	7.0
8 ピクセル (150 MHz)	1.87	1.87

## 複製 (duplicateMat)

パイプラインのさまざまな関数がプログラマブルロジックでインプリメントされる場合、データフロー処理のため、2つの関数の間に FIFO がインスタンス化されます。1つの関数からの出力がパイプラインの2つの関数で使われる場合、FIFO を複製する必要があります。この関数は、FIFO の複製プロセスを簡単に実行できるようにします。

### API 構文

```
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)
```

### パラメーターの説明

次の表に、テンプレートと関数のパラメーターを説明します。

パラメーター	説明
SRC_T	入力および出力のピクセルのデータ型
ROWS	入力および出力画像の最大高さ。
COLS	入力および出力画像の最大幅 (8 ピクセル動作の場合は 8 の倍数で指定)
NPC	1 サイクルごとに処理されるピクセル数。使用可能なオプションは、1 ピクセルの場合は XF_NPPC1、8 ピクセルの場合は XF_NPPC8。
_src	入力画像
_dst1	_src の複製出力

パラメーター	説明
_dst2	_src の複製出力

## 収縮 (erode)

erode 関数は、ピクセルの近傍 NXN で最小のピクセル強度を見つけ、ピクセル強度をその最小値に置き換えます。

$$dst(x, y) = \min_{\substack{x-1 \leq x' \leq x+1 \\ y-1 \leq y' \leq y+1}} src(x', y')$$

### API 構文

```
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])
```

### パラメーターの説明

次の表に、テンプレートと関数のパラメーターを説明します。

表 231: erode 関数のパラメーターの説明

パラメーター	説明
BORDER_TYPE	サポートされる境界タイプは XF_BORDER_CONSTANT。
TYPE	入力および出力のピクセルのデータ型。8 ビット、符号なし、1 および 3 チャンネル (XF_8UC1、XF_8UC3) のみサポート。
ROWS	入力および出力画像の最大高さ。
COLS	入力および出力画像の最大幅 (8 ピクセル動作の場合は 8 の倍数で指定)
K_SHAPE	カーネルの形状。サポートされるカーネルの形状は RECT、CROSS、および ELLIPSE です。
K_ROWS	カーネルの高さ。
K_COLS	カーネルの幅。
ITERATIONS	収縮の適用回数。現在のところ、矩形 (RECT) のカーネル要素でのみサポートされます。
NPC	1 サイクルごとに処理されるピクセル数。1 ピクセルの場合は XF_NPPC1、8 ピクセルの場合は XF_NPPC8。
_src_mat	入力画像
_dst_mat	出力画像
_kernel	サイズ K_ROWS * K_COLS の収縮カーネル。

### リソース使用量

次の表に、Xczu9eg-ffvb1156-1-i-es1 FPGA でフル HD 画像 (1080x1920) を処理するために、Vivado HLS 2019.1 バージョン ツールで矩形 (RECT) 構造要素を使用して生成された収縮関数のリソース使用量を示します。

表 232: erode 関数のリソース使用量のサマリ

名前	リソース使用量	
	1 クロックごとに 1 ピクセル動作	8 クロックごとに 1 ピクセル動作
	300 MHz	150 MHz
BRAM_18K	3	6
DSP48E	0	0
FF	411	657
LUT	392	1249
CLB	96	255

次の表に、Xczu9eg-ffvb1156-1-i-es1 FPGA で 3 チャンネルの 4K 画像を処理するために、Vivado HLS 2019.1 バージョン ツールで矩形 (RECT) 構造要素を使用して生成された収縮関数のリソース使用量を示します。

表 233: erode 関数のリソース使用量のサマリ

名前	リソース使用量
	1 クロックごとに 1 ピクセル動作
	300 MHz
BRAM_18K	18
DSP48E	0
FF	983
LUT	3745
CLB	186

### パフォーマンス見積もり

次の表に、Xczu9eg-ffvb1156-1-i-es1 FPGA で Vivado HLS 2019.1 ツールを使用して生成された、通常動作 (1 ピクセル) およびリソース最適化 (8 ピクセル) 設定での収縮関数のパフォーマンス見積もりを示します。

表 234: erode 関数のパフォーマンス見積もりのサマリ

動作モード	レイテンシ見積もり	
	最小 (ms)	最大 (ms)
1 ピクセル (300 MHz)	7.0	7.0
8 ピクセル (150 MHz)	1.85	1.85

## FAST コーナー検出 (fast)

FAST (Features from Accelerated Segment Test) は、ほかのほとんどの特徴検出より高速のコーナー検出アルゴリズムです。

`fast` 関数は、画像内のピクセルを 1 つ選び、Bresenham の円と呼ばれる円上の近傍ピクセル 16 個の強度を比較します。9 個の連続するピクセルの強度が候補のピクセルよりも指定のしきい値分大きい小さい場合、ピクセルをコーナーとみなします。コーナーが検出されると NMS (Non-Maximal Suppression) が適用され、弱いコーナーが削除されます。

この関数は、静止画像およびビデオの両方に使用できます。コーナーは画像でマークされます。コーナーが特定の場所で検出された場合、その場所は 255 となり、それ以外の場所は 0 になります。

## API 構文

```
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)
```

## パラメーターの説明

次の表に、テンプレートと関数のパラメーターを説明します。

表 235: fast 関数のパラメーターの説明

パラメーター	説明
NMS	NMS == 1 の場合、検出されたコーナー (キーポイント) に NMS が適用されます。有効な値は 0 または 1 です。
SRC_T	入力ピクセルのデータ型。8 ビット、符号なし、1 チャンネル (XF_8UC1) のみサポート。
ROWS	入力画像の最大高さ。
COLS	入力画像の最大幅 (8 ピクセル動作の場合は 8 の倍数で指定)
NPC	1 サイクルごとに処理されるピクセル数。1 ピクセルの場合は XF_NPPC1、8 ピクセルの場合は XF_NPPC8。
_src_mat	入力画像
_dst_mat	出力画像。コーナーは画像でマークされます。
_threshold	中央ピクセルと近傍ピクセルとの強度の差のしきい値。通常 20 前後の値が使用されます。

## リソース使用量

次の表に、Xczu9eg-ffvb1156-1-i-es1 FPGA でグレースケール HD (1080x1920) 画像を NMS を使用して処理するために、Vivado HLS 2019.1 ツールを使用して生成された異なる設定でのカーネルのリソース使用量を示します。

表 236: fast 関数のリソース使用量のサマリ

名前	リソース使用量	
	1 ピクセル	8 ピクセル
	300 MHz	150 MHz
BRAM_18K	10	20
DSP48E	0	0
FF	2695	7310
LUT	3792	20956
CLB	769	3519

## パフォーマンス見積もり

次の表に、Xczu9eg-ffvb1156-1-i-es1 でグレースケール HD (1080x1920) 画像を NMS を使用して処理するために、Vivado HLS 2019.1 ツールを使用して生成された異なる設定でのカーネルのパフォーマンス見積もりを示します。



表 237: fast 関数のパフォーマンス見積もりのサマリ

動作モード	動作周波数 (MHz)	フィルター サイズ	レイテンシ見積もり
			最大 (ms)
1 ピクセル	300	3x3	7
8 ピクセル	150	3x3	1.86

## ガウシアン フィルター (GaussianBlur)

GaussianBlur 関数は、入力画像にガウシアンぼかしを適用します。ガウシアンフィルター処理は、入力画像の各点をガウシアン カーネルでたたみ込むことにより実行されます。

$$G_0(x, y) = e^{-\frac{(x - \mu_x)^2}{2\sigma_x^2} - \frac{(y - \mu_y)^2}{2\sigma_y^2}}$$

ここで、 $\mu_x$  および  $\mu_y$  は平均値、 $\sigma_x$  および  $\sigma_y$  はそれぞれ x 方向と y 方向の分散です。GaussianBlur 関数では、 $\mu_x$  および  $\mu_y$  は 0、 $\sigma_x$  および  $\sigma_y$  は等しいと考慮されます。

### API 構文

```
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)
```

### パラメーターの説明

次の表に、テンプレートと関数のパラメーターを説明します。

表 238: GaussianBlur 関数のパラメーターの説明

パラメーター	説明
FILTER_SIZE	フィルター サイズ。サポートされるフィルター サイズは 3 (XF_FILTER_3X3)、5 (XF_FILTER_5X5)、および 7 (XF_FILTER_7X7)。
BORDER_TYPE	サポートされる境界タイプは XF_BORDER_CONSTANT。
SRC_T	入力および出力のピクセルのデータ型。8 ビット、符号なし、1 および 3 チャンネル (XF_8UC1、XF_8UC3) のみサポート。
ROWS	入力および出力画像の最大高さ。
COLS	入力および出力画像の最大幅 (8 ピクセル動作の場合は 8 の倍数で指定)
NPC	1 サイクルごとに処理されるピクセル数。1 ピクセルの場合は XF_NPPC1、8 ピクセルの場合は XF_NPPC8。
src	入力画像
dst	出力画像
sigma	ガウシアン フィルターの標準偏差

## リソース使用量

次の表に、Xczu9eg-ffvb1156-1-i-es1 FPGA でグレースケール HD (1080x1920) 画像を処理するために、Vivado HLS 2019.1 ツールを使用して生成された異なる設定でのガウシアンフィルターのリソース使用量を示します。

表 239: GaussianBlur 関数のリソース使用量のサマリ

動作モード	フィルター サイズ	動作周波数 (MHz)	使用量の見積もり				
			BRAM_18K	DSP_48E	FF	LUT	CLB
1 ピクセル	3x3	300	3	17	3641	2791	610
	5x5	300	5	27	4461	3544	764
	7x7	250	7	35	4770	4201	894
8 ピクセル	3x3	150	6	52	3939	3784	814
	5x5	150	10	111	5688	5639	1133
	7x7	150	14	175	7594	7278	1518

次の表に、Xczu9eg-ffvb1156-1-i-es1 FPGA で 3 チャンネルの 4K 画像を処理するために、Vivado HLS 2019.1 ツールを使用して生成された異なる設定でのガウシアン フィルターのリソース使用量を示します。

表 240: GaussianBlur 関数のリソース使用量のサマリ

動作モード	フィルター サイズ	動作周波数 (MHz)	使用量の見積もり			
			BRAM_18K	DSP_48E	FF	LUT
1 ピクセル	3x3	300	18	33	4835	3472
	5x5	300	30	51	5755	3994
	7x7	300	42	135	8086	5422

## パフォーマンス見積もり

次の表に、Xczu9eg-ffvb1156-1-i-es1 FPGA でグレースケール HD (1080x1920) 画像を処理するために、Vivado HLS 2019.1 ツールを使用して生成された異なる設定でのガウシアン フィルターのパフォーマンス見積もりを示します。

表 241: GaussianBlur 関数のパフォーマンス見積もりのサマリ

動作モード	フィルター サイズ	レイテンシ見積もり
		最大レイテンシ (ms)
1 ピクセル動作 (300 MHz)	3x3	7.01
	5x5	7.03
	7x7	7.06
8 ピクセル動作 (150 MHz)	3x3	1.6
	5x5	1.7
	7x7	1.74

## 勾配の大きさ (magnitude)

`magnitude` 関数では、画像の大きさが計算されます。入力画像は、16S 型の X 勾配および Y 勾配画像です。出力画像は、入力画像と同型になります。

L1NORM 正規化の場合、大きさの計算された画像は、次のように X 勾配と Y 勾配の絶対値のピクセル加算された画像になります。

$$g = |g_x| + |g_y|$$

L2NORM 正規化の場合、画像の大きさは次のように計算されます。

$$g = \sqrt{(g_x^2 + g_y^2)}$$

### API 構文

```
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)
```

### パラメーターの説明

次の表に、テンプレートと関数のパラメーターを説明します。

表 242: `magnitude` 関数のパラメーターの説明

パラメーター	説明
NORM_TYPE	正規型は L1 または L2 正規化にでき、値は XF_L1NORM または XF_L2NORM です。
SRC_T	入力ピクセルのデータ型。16 ビット、符号付き、1 チャンネル (XF_16SC1) のみサポート。
DST_T	出力ピクセルのデータ型。16 ビット、符号付き、1 チャンネル (XF_16SC1) のみサポート。
ROWS	入力および出力画像の最大高さ。
COLS	入力および出力画像の最大幅 (8 ピクセル動作の場合は 8 の倍数で指定)
NPC	1 サイクルごとに処理されるピクセル数。1 ピクセルの場合は XF_NPPC1、8 ピクセルの場合は XF_NPPC8。
_src_matx	最初の入力、X 勾配画像。
_src_maty	2 つ目の入力、Y 勾配画像。
_dst_mat	出力、大きさの計算された画像。

### リソース使用量

次の表に、Xczu9eg-ffvb1156-1-i-es1 FPGA でグレースケール HD (1080x1920) 画像で L2 正規化の処理をするために、Vivado HLS 2019.1 ツールを使用して生成された異なる設定でのカーネルのリソース使用量を示します。

表 243: magnitude 関数のリソース使用量のサマリ

名前	リソース使用量	
	1 ピクセル	8 ピクセル
	300 MHz	150 MHz
BRAM_18K	0	0
DSP48E	2	16
FF	707	2002
LUT	774	3666
CLB	172	737

### パフォーマンス見積もり

次の表に、Xczu9eg-ffvb1156-1-i-es1 でグレースケール HD (1080x1920) 画像で L2 正規化の処理をするために、Vivado HLS 2019.1 ツールを使用して生成された異なる設定でのカーネルのパフォーマンス見積もりを示します。

表 244: magnitude 関数のパフォーマンス見積もりのサマリ

動作モード	動作周波数 (MHz)	レイテンシ見積もり
		最大 (ms)
1 ピクセル	300	7.2
8 ピクセル	150	1.7

## 勾配位相 (phase)

phase 関数では、2 つの画像の極角が計算されます。入力画像は、16S 型の X 勾配および Y 勾配画像です。出力画像は、入力画像と同型になります。

ラジアン (radians) の場合:

$$angle(x, y) = atan2(g_y, g_x)$$

度 (degrees) の場合:

$$angle(x, y) = atan2(g_y, g_x) * \frac{180}{\pi}$$

### API 構文

```
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)
```

### パラメーターの説明

次の表に、テンプレートと関数のパラメーターを説明します。

表 245: phase 関数のパラメーターの説明

パラメーター	説明
RET_TYPE	出力フォーマットは、ラジアン (radians) または度 (degrees) のいずれかにできます。オプションは XF_RADIANS または XF_DEGREES です。 <ul style="list-style-type: none"> <li>XF_RADIANS: Q4.12 フォーマットの結果が返されます。出力範囲は (0, 2 pi) です。</li> <li>XF_DEGREES: 結果が Q10.6 度で返されます。出力範囲は (0, 360) です。</li> </ul>
SRC_T	入力ピクセルのデータ型。16 ビット、符号付き、1 チャンネル (XF_16SC1) のみサポート。
DST_T	出力ピクセルのデータ型。16 ビット、符号付き、1 チャンネル (XF_16SC1) のみサポート。
ROWS	入力および出力画像の最大高さ。
COLS	入力および出力画像の最大幅 (8 ピクセル動作の場合は 8 の倍数で指定)
NPC	1 サイクルごとに処理されるピクセル数。1 ピクセルの場合は XF_NPPC1、8 ピクセルの場合は XF_NPPC8。
_src_matx	最初の入力、X 勾配画像。
_src_maty	2 つ目の入力、Y 勾配画像。
_dst_mat	出力、位相計算された画像。

## リソース使用量

次の表に、Xczu9eg-ffvb1156-1-i-es1 FPGA でグレースケール HD (1080x1920) 画像を処理するために、Vivado HLS 2019.1 ツールを使用して生成された異なる設定でのカーネルのリソース使用量を示します。

表 246: phase 関数のリソース使用量のサマリ

名前	リソース使用量	
	1 ピクセル	8 ピクセル
	300 MHz	150 MHz
BRAM_18K	6	24
DSP48E	6	19
FF	873	2396
LUT	753	3895
CLB	185	832

## パフォーマンス見積もり

次の表に、Xczu9eg-ffvb1156-1-i-es1 でグレースケール HD (1080x1920) 画像を処理するために、Vivado HLS 2019.1 ツールを使用して生成された異なる設定でのカーネルのパフォーマンス見積もりを示します。

表 247: phase 関数のパフォーマンス見積もりのサマリ

動作モード	動作周波数 (MHz)	レイテンシ見積もり (ms)
1 ピクセル	300	7.2
8 ピクセル	150	1.7

## OpenCV との違い

phase インプリメンテーションでは、出力が固定小数点フォーマットで返されます。XF\_RADIANS オプションを選択すると、結果が Q4.12 フォーマットで返されます。出力範囲は (0, 2 pi) です。XF\_DEGREES オプションを選択すると、結果が Q10.6 度で返されます。出力範囲は (0, 360) です。

## Harris コーナー検出 (cornerHarris)

Harris コーナー検出を理解するには、グレースケール画像を考慮します。ウィンドウ  $w(x, y)$  (x 方向に  $u$ 、y 方向に  $v$  移動) をスライプし、 $I$  で  $w(x, y)$  の強度の変動を計算します。

$$E(u, v) = \sum w(x, y) [I(x + u, y + v) - I(x, y)]^2$$

説明:

- $w(x, y)$ : (x,y) のウィンドウ位置。
- $I(x, y)$ : (x,y) での強度。
- $I(x+u, y+v)$ : 移動したウィンドウ (x+u, y+v) での強度。

コーナーのあるウィンドウを探しているので、強度の変動が大きいウィンドウを見つけます。上記の式、特に次の項を最大化する必要があります。

$$[I(x + u, y + v) - I(x, y)]^2$$

テーラー展開を使用します。

$$E(u, v) = \sum [I(x, y) + uI_x + vI_y - I(x, y)]^2$$

式を展開して  $I(x, y)$  を  $-I(x, y)$  で相殺します。

$$E(u, v) = \sum u^2 I_x^2 + 2uv I_x I_y + v^2 I_y^2$$

上記の式を行列で表すと、次のようになります。

$$E(u, v) = [u \ v] \left( \sum w(x, y) \begin{bmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{bmatrix} \right) \begin{bmatrix} u \\ v \end{bmatrix}$$

最終的な式は、次のようになります。

$$E(u, v) = [u \ v] M \begin{bmatrix} u \\ v \end{bmatrix}$$

各ウィンドウに対してスコアが計算され、コーナーを含むかどうか判断されます。

$$R = \det(M) - k(\text{trace}(M))^2$$

説明:

- $\det(M) = \lambda_1 \lambda_2$
- $\text{trace}(M) = \lambda_1 + \lambda_2$

NMS (Non-Maximum Suppression):

NMS では、範囲が 1 の場合境界ボックスは  $2*r+1=3$  です。

中央ピクセルの近傍 3x3 ピクセルを考慮します。中央ピクセルが周辺ピクセルより大きい場合、これはコーナーと考えられます。範囲内の周辺ピクセルと比較されます。

範囲 = 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

しきい値:

3x3、5x5、および 7x7 フィルターに対して、それぞれしきい値 442、3109、および 566 が使用されます。このしきい値は、40 を超える画像のセットで検証されます。しきい値は、アプリケーションによって異なります。コーナーは出力画像でマークされます。コーナーが特定の場所で検出された場合、その場所は 255 となり、それ以外の場所は 0 になります。

## API 構文

```
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)
```

## パラメーターの説明

次の表に、テンプレートと関数のパラメーターを説明します。

表 248: cornerHarris 関数のパラメーターの説明

パラメーター	説明
FILTERSIZE	Sobel フィルターのサイズ。有効な値は 3、5、7。
BLOCKWIDTH	ボックス フィルターのサイズ。有効な値は 3、5、7。
NMSRADIUS	NMS で考慮される範囲。有効な値は 1 および 2。
TYPE	入力ピクセルのデータ型。8 ビット、符号なし、1 チャンネル (XF_8UC1) のみサポート。
ROWS	入力画像の最大高さ。
COLS	入力画像の最大幅 (8 ピクセル動作の場合は 8 の倍数で指定)
NPC	1 サイクルごとに処理されるピクセル数。1 ピクセルの場合は XF_NPPC1、8 ピクセルの場合は XF_NPPC8。
USE_URAM	一部のストレージ構造を UltraRAM にマップ
src	入力画像
dst	出力画像。
threshold	コーナー計測に適用されるしきい値。

表 248: cornerHarris 関数のパラメーターの説明 (続き)

パラメーター	説明
k	Harris 検出器パラメーター

### リソース使用量

次の表に、Xczu9eg-ffvb1156-1-i-es1 FPGA でグレースケール HD (1080x1920) 画像を処理するために、Vivado HLS 2019.1 ツールを使用して生成された異なる設定での Harris コーナー検出のリソース使用量を示します。

次の表に、Sobel フィルター = 3、ボックス フィルター = 3、NMS\_RADIUS = 1 のリソース使用量を示します。

表 249: リソース使用量のサマリ: Sobel フィルター = 3、ボックス フィルター = 3、NMS\_RADIUS = 1

名前	リソース使用量	
	1 ピクセル	8 ピクセル
	300 MHz	150 MHz
BRAM_18K	33	66
DSP48E	10	80
FF	3254	9330
LUT	3522	13222
CLB	731	2568

次の表に、Sobel フィルター = 3、ボックス フィルター = 5、NMS\_RADIUS = 1 のリソース使用量を示します。

表 250: リソース使用量のサマリ: Sobel フィルター = 3、ボックス フィルター = 5、NMS\_RADIUS = 1

名前	リソース使用量	
	1 ピクセル	8 ピクセル
	300 MHz	150 MHz
BRAM_18K	45	90
DSP48E	10	80
FF	5455	12459
LUT	5675	24594
CLB	1132	4498

次の表に、Sobel フィルター = 3、ボックス フィルター = 7、NMS\_RADIUS = 1 のリソース使用量を示します。

表 251: リソース使用量のサマリ: Sobel フィルター = 3、ボックス フィルター = 7、NMS\_RADIUS = 1

名前	リソース使用量	
	1 ピクセル	8 ピクセル
	300 MHz	150 MHz
BRAM_18K	57	114
DSP48E	10	80
FF	8783	16593



表 251: リソース使用量のサマリ: Sobel フィルター=3、ボックス フィルター=7、NMS\_RADIUS=1 (続き)

名前	リソース使用量	
	1 ピクセル	8 ピクセル
	300 MHz	150 MHz
LUT	9157	39813
CLB	1757	6809

次の表に、Sobel フィルター=5、ボックス フィルター=3、NMS\_RADIUS=1 のリソース使用量を示します。

表 252: リソース使用量のサマリ: Sobel フィルター=5、ボックス フィルター=3、NMS\_RADIUS=1

名前	リソース使用量	
	1 ピクセル	8 ピクセル
	300 MHz	200 MHz
BRAM_18K	35	70
DSP48E	10	80
FF	4656	11659
LUT	4681	17394
CLB	1005	3277

次の表に、Sobel フィルター=5、ボックス フィルター=5、NMS\_RADIUS=1 のリソース使用量を示します。

表 253: リソース使用量のサマリ: Sobel フィルター=5、ボックス フィルター=5、NMS\_RADIUS=1

名前	リソース使用量	
	1 ピクセル	8 ピクセル
	300 MHz	150 MHz
BRAM_18K	47	94
DSP48E	10	80
FF	6019	14776
LUT	6337	28795
CLB	1353	5102

次の表に、Sobel フィルター=5、ボックス フィルター=7、NMS\_RADIUS=1 のリソース使用量を示します。

表 254: リソース使用量のサマリ: Sobel フィルター=5、ボックス フィルター=7、NMS\_RADIUS=1

名前	リソース使用量	
	1 ピクセル	8 ピクセル
	300 MHz	150 MHz
BRAM_18K	59	118
DSP48E	10	80
FF	9388	18913
LUT	9414	43070

表 254: リソース使用量のサマリ: Sobel フィルター = 5、ボックス フィルター = 7、NMS\_RADIUS = 1 (続き)

名前	リソース使用量	
	1 ピクセル	8 ピクセル
	300 MHz	150 MHz
CLB	1947	7508

次の表に、Sobel フィルター = 7、ボックス フィルター = 3、NMS\_RADIUS = 1 のリソース使用量を示します。

表 255: リソース使用量のサマリ: Sobel フィルター = 7、ボックス フィルター = 3、NMS\_RADIUS = 1

名前	リソース使用量	
	1 ピクセル	8 ピクセル
	300 MHz	150 MHz
BRAM_18K	37	74
DSP48E	11	88
FF	6002	13880
LUT	6337	25573
CLB	1327	4868

次の表に、Sobel フィルター = 7、ボックス フィルター = 5、NMS\_RADIUS = 1 のリソース使用量を示します。

表 256: リソース使用量のサマリ: Sobel フィルター = 7、ボックス フィルター = 5、NMS\_RADIUS = 1

名前	リソース使用量	
	1 ピクセル	8 ピクセル
	300 MHz	150 MHz
BRAM_18K	49	98
DSP48E	11	88
FF	7410	17049
LUT	8076	36509
CLB	1627	6518

次の表に、Sobel フィルター = 7、ボックス フィルター = 7、NMS\_RADIUS = 1 のリソース使用量を示します。

表 257: リソース使用量のサマリ: Sobel フィルター = 7、ボックス フィルター = 7、NMS\_RADIUS = 1

名前	リソース使用量	
	1 ピクセル	8 ピクセル
	300 MHz	150 MHz
BRAM_18K	61	122
DSP48E	11	88
FF	10714	21137
LUT	11500	51331
CLB	2261	8863

次の表に、Sobel フィルター = 3、ボックス フィルター = 3、NMS\_RADIUS = 2 のリソース使用量を示します。

表 258: リソース使用量のサマリ: Sobel フィルター = 3、ボックス フィルター = 3、NMS\_RADIUS = 2

名前	リソース使用量	
	1 ピクセル	8 ピクセル
	300 MHz	150 MHz
BRAM_18K	41	82
DSP48E	10	80
FF	5519	10714
LUT	5094	16930
CLB	1076	3127

次の表に、Sobel フィルター = 3、ボックス フィルター = 5、NMS\_RADIUS = 2 のリソース使用量を示します。

表 259: リソース使用量のサマリ

名前	リソース使用量	
	1 ピクセル	8 ピクセル
	300 MHz	150 MHz
BRAM_18K	53	106
DSP48E	10	80
FF	6798	13844
LUT	6866	28286
CLB	1383	4965

次の表に、Sobel フィルター = 3、ボックス フィルター = 7、NMS\_RADIUS = 2 のリソース使用量を示します。

表 260: リソース使用量のサマリ: Sobel フィルター = 3、ボックス フィルター = 7、NMS\_RADIUS = 2

名前	リソース使用量	
	1 ピクセル	8 ピクセル
	300 MHz	150 MHz
BRAM_18K	65	130
DSP48E	10	80
FF	10137	17977
LUT	10366	43589
CLB	1940	7440

次の表に、Sobel フィルター = 5、ボックス フィルター = 3、NMS\_RADIUS = 2 のリソース使用量を示します。

表 261: リソース使用量のサマリ: Sobel フィルター=5、ボックス フィルター=3、NMS\_RADIUS=2

名前	リソース使用量	
	1 ピクセル	8 ピクセル
	300 MHz	150 MHz
BRAM_18K	43	86
DSP48E	10	80
FF	5957	12930
LUT	5987	21187
CLB	1244	3922

次の表に、Sobel フィルター=5、ボックス フィルター=5、NMS\_RADIUS=2 のリソース使用量を示します。

表 262: リソース使用量のサマリ: Sobel フィルター=5、ボックス フィルター=5、NMS\_RADIUS=2

名前	リソース使用量	
	1 ピクセル	8 ピクセル
	300 MHz	150 MHz
BRAM_18K	55	110
DSP48E	10	80
FF	5442	16053
LUT	6561	32377
CLB	1374	5871

次の表に、Sobel フィルター=5、ボックス フィルター=7、NMS\_RADIUS=2 のリソース使用量を示します。

表 263: リソース使用量のサマリ: Sobel フィルター=5、ボックス フィルター=7、NMS\_RADIUS=2

名前	リソース使用量	
	1 ピクセル	8 ピクセル
	300 MHz	150 MHz
BRAM_18K	67	134
DSP48E	10	80
FF	10673	20190
LUT	10793	46785
CLB	2260	8013

次の表に、Sobel フィルター=7、ボックス フィルター=3、NMS\_RADIUS=2 のリソース使用量を示します。

表 264: リソース使用量のサマリ: Sobel フィルター=7、ボックス フィルター=3、NMS\_RADIUS=2

名前	リソース使用量	
	1 ピクセル	8 ピクセル
	300 MHz	150 MHz
BRAM_18K	45	90

表 264: リソース使用量のサマリ: Sobel フィルター=7、ボックス フィルター=3、NMS\_RADIUS=2 (続き)

名前	リソース使用量	
	1 ピクセル	8 ピクセル
	300 MHz	150 MHz
DSP48E	11	88
FF	7341	15161
LUT	7631	29185
CLB	1557	5425

次の表に、Sobel フィルター=7、ボックス フィルター=5、NMS\_RADIUS=2 のリソース使用量を示します。

表 265: リソース使用量のサマリ: Sobel フィルター=7、ボックス フィルター=5、NMS\_RADIUS=2

名前	リソース使用量	
	1 ピクセル	8 ピクセル
	300 MHz	150 MHz
BRAM_18K	57	114
DSP48E	11	88
FF	8763	18330
LUT	9368	40116
CLB	1857	7362

次の表に、Sobel フィルター=7、ボックス フィルター=7、NMS\_RADIUS=2 のリソース使用量を示します。

表 266: リソース使用量のサマリ: Sobel フィルター=7、ボックス フィルター=7、NMS\_RADIUS=2

名前	リソース使用量	
	1 ピクセル	8 ピクセル
	300 MHz	150 MHz
BRAM_18K	69	138
DSP48E	11	88
FF	12078	22414
LUT	12831	54652
CLB	2499	9628

#### URAM をイネーブルにした場合のリソース使用量

次の表に、xczu7ev-ffvc1156-2-e FPGA でグレースケール 4K (3840X2160) 画像を処理するために、SDx 2019.1 ツールを使用して生成された異なる設定での Harris コーナー検出のリソース使用量を示します。

次の表に、Sobel フィルター=3、ボックス フィルター=3、NMS\_RADIUS=1 のリソース使用量を示します。

表 267: リソース使用量のサマリ: Sobel フィルター=3、ボックス フィルター=3、NMS\_RADIUS=1

名前	リソース使用量	
	1 ピクセル	8 ピクセル
	300 MHz	150 MHz
BRAM_18K	12	12
URAM	4	21
DSP48E	10	80
FF	5306	11846
LUT	3696	13846

次の表に、Sobel フィルター=3、ボックス フィルター=5、NMS\_RADIUS=1 のリソース使用量を示します。

表 268: リソース使用量のサマリ: Sobel フィルター=3、ボックス フィルター=5、NMS\_RADIUS=1

名前	リソース使用量	
	1 ピクセル	8 ピクセル
	300 MHz	150 MHz
BRAM_18K	12	12
URAM	7	30
DSP48E	10	80
FF	7625	13899
LUT	5596	27136

次の表に、Sobel フィルター=3、ボックス フィルター=7、NMS\_RADIUS=1 のリソース使用量を示します。

表 269: リソース使用量のサマリ: Sobel フィルター=3、ボックス フィルター=7、NMS\_RADIUS=1

名前	リソース使用量	
	1 ピクセル	8 ピクセル
	300 MHz	150 MHz
BRAM_18K	12	12
URAM	7	42
DSP48E	10	80
FF	12563	19919
LUT	8816	39087

次の表に、Sobel フィルター=5、ボックス フィルター=3、NMS\_RADIUS=1 のリソース使用量を示します。

表 270: リソース使用量のサマリ: Sobel フィルター=5、ボックス フィルター=3、NMS\_RADIUS=1

名前	リソース使用量	
	1 ピクセル	8 ピクセル
	300 MHz	150 MHz
BRAM_18K	12	12

表 270: リソース使用量のサマリ: Sobel フィルター=5、ボックス フィルター=3、NMS\_RADIUS=1 (続き)

名前	リソース使用量	
	1 ピクセル	8 ピクセル
	300 MHz	150 MHz
URAM	4	23
DSP48E	10	80
FF	6689	15022
LUT	4506	18719

次の表に、Sobel フィルター=5、ボックス フィルター=5、NMS\_RADIUS=1 のリソース使用量を示します。

表 271: リソース使用量のサマリ: Sobel フィルター=5、ボックス フィルター=5、NMS\_RADIUS=1

名前	リソース使用量	
	1 ピクセル	8 ピクセル
	300 MHz	150 MHz
BRAM_18K	12	12
URAM	7	32
DSP48E	10	80
FF	9050	17063
LUT	6405	31992

次の表に、Sobel フィルター=5、ボックス フィルター=7、NMS\_RADIUS=1 のリソース使用量を示します。

表 272: リソース使用量のサマリ: Sobel フィルター=5、ボックス フィルター=7、NMS\_RADIUS=1

名前	リソース使用量	
	1 ピクセル	8 ピクセル
	300 MHz	150 MHz
BRAM_18K	12	12
URAM	7	44
DSP48E	10	80
FF	13946	23116
LUT	9626	44738

次の表に、Sobel フィルター=7、ボックス フィルター=3、NMS\_RADIUS=1 のリソース使用量を示します。

表 273: リソース使用量のサマリ: Sobel フィルター=7、ボックス フィルター=3、NMS\_RADIUS=1

名前	リソース使用量	
	1 ピクセル	8 ピクセル
	300 MHz	150 MHz
BRAM_18K	12	12
URAM	4	25

表 273: リソース使用量のサマリ: Sobel フィルター=7、ボックス フィルター=3、NMS\_RADIUS=1 (続き)

名前	リソース使用量	
	1 ピクセル	8 ピクセル
	300 MHz	150 MHz
DSP48E	11	88
FF	8338	17378
LUT	6151	24844

次の表に、Sobel フィルター=7、ボックス フィルター=5、NMS\_RADIUS=1 のリソース使用量を示します。

表 274: リソース使用量のサマリ: Sobel フィルター=7、ボックス フィルター=5、NMS\_RADIUS=1

名前	リソース使用量	
	1 ピクセル	8 ピクセル
	300 MHz	150 MHz
BRAM_18K	12	12
URAM	7	34
DSP48E	11	88
FF	10497	19457
LUT	7858	39762

次の表に、Sobel フィルター=7、ボックス フィルター=7、NMS\_RADIUS=1 のリソース使用量を示します。

表 275: リソース使用量のサマリ: Sobel フィルター=7、ボックス フィルター=7、NMS\_RADIUS=1

名前	リソース使用量	
	1 ピクセル	8 ピクセル
	300 MHz	150 MHz
BRAM_18K	12	12
URAM	7	46
DSP48E	11	88
FF	15393	25450
LUT	11080	50662

次の表に、Sobel フィルター=3、ボックス フィルター=3、NMS\_RADIUS=2 のリソース使用量を示します。

表 276: リソース使用量のサマリ: Sobel フィルター=3、ボックス フィルター=3、NMS\_RADIUS=2

名前	リソース使用量	
	1 ピクセル	8 ピクセル
	300 MHz	150 MHz
BRAM_18K	20	20
URAM	4	21
DSP48E	10	80



表 276: リソース使用量のサマリ: Sobel フィルター=3、ボックス フィルター=3、NMS\_RADIUS=2 (続き)

名前	リソース使用量	
	1 ピクセル	8 ピクセル
	300 MHz	150 MHz
FF	6286	13441
LUT	4704	18072

次の表に、Sobel フィルター=3、ボックス フィルター=5、NMS\_RADIUS=2 のリソース使用量を示します。

表 277: リソース使用量のサマリ

名前	リソース使用量	
	1 ピクセル	8 ピクセル
	300 MHz	150 MHz
BRAM_18K	20	20
URAM	7	30
DSP48E	10	80
FF	8626	15498
LUT	6606	31371

次の表に、Sobel フィルター=3、ボックス フィルター=7、NMS\_RADIUS=2 のリソース使用量を示します。

表 278: リソース使用量のサマリ: Sobel フィルター=3、ボックス フィルター=7、NMS\_RADIUS=2

名前	リソース使用量	
	1 ピクセル	8 ピクセル
	300 MHz	150 MHz
BRAM_18K	20	20
URAM	7	42
DSP48E	10	80
FF	13543	21522
LUT	9853	43301

次の表に、Sobel フィルター=5、ボックス フィルター=3、NMS\_RADIUS=2 のリソース使用量を示します。

表 279: リソース使用量のサマリ: Sobel フィルター=5、ボックス フィルター=3、NMS\_RADIUS=2

名前	リソース使用量	
	1 ピクセル	8 ピクセル
	300 MHz	150 MHz
BRAM_18K	20	20
URAM	4	23
DSP48E	10	80
FF	7670	16750

表 279: リソース使用量のサマリ: Sobel フィルター=5、ボックス フィルター=3、NMS\_RADIUS=2 (続き)

名前	リソース使用量	
	1 ピクセル	8 ピクセル
	300 MHz	150 MHz
LUT	5513	22854

次の表に、Sobel フィルター=5、ボックス フィルター=5、NMS\_RADIUS=2 のリソース使用量を示します。

表 280: リソース使用量のサマリ: Sobel フィルター=5、ボックス フィルター=5、NMS\_RADIUS=2

名前	リソース使用量	
	1 ピクセル	8 ピクセル
	300 MHz	150 MHz
BRAM_18K	20	20
URAM	7	32
DSP48E	10	80
FF	9712	18793
LUT	7338	36136

次の表に、Sobel フィルター=5、ボックス フィルター=7、NMS\_RADIUS=2 のリソース使用量を示します。

表 281: リソース使用量のサマリ: Sobel フィルター=5、ボックス フィルター=7、NMS\_RADIUS=2

名前	リソース使用量	
	1 ピクセル	8 ピクセル
	300 MHz	150 MHz
BRAM_18K	20	20
URAM	7	44
DSP48E	10	80
FF	14650	24846
LUT	10558	48866

次の表に、Sobel フィルター=7、ボックス フィルター=3、NMS\_RADIUS=2 のリソース使用量を示します。

表 282: リソース使用量のサマリ: Sobel フィルター=7、ボックス フィルター=3、NMS\_RADIUS=2

名前	リソース使用量	
	1 ピクセル	8 ピクセル
	300 MHz	150 MHz
BRAM_18K	20	20
URAM	4	25
DSP48E	11	88
FF	9562	19101
LUT	7405	29986

次の表に、Sobel フィルター=7、ボックス フィルター=5、NMS\_RADIUS=2 のリソース使用量を示します。

表 283: リソース使用量のサマリ: Sobel フィルター=7、ボックス フィルター=5、NMS\_RADIUS=2

名前	リソース使用量	
	1 ピクセル	8 ピクセル
	300 MHz	150 MHz
BRAM_18K	20	20
URAM	7	34
DSP48E	11	88
FF	11751	21180
LUT	9254	44024

次の表に、Sobel フィルター=7、ボックス フィルター=7、NMS\_RADIUS=2 のリソース使用量を示します。

表 284: リソース使用量のサマリ: Sobel フィルター=7、ボックス フィルター=7、NMS\_RADIUS=2

名前	リソース使用量	
	1 ピクセル	8 ピクセル
	300 MHz	150 MHz
BRAM_18K	20	20
URAM	7	46
DSP48E	11	88
FF	16723	27156
LUT	12474	54858

## パフォーマンス見積もり

次の表に、ザイリンクス Xczu9eg-ffvb1156-1-i-es1 FPGA でグレースケール HD (1080x1920) 画像を処理するために、Vivado HLS 2019.1 ツールを使用して生成された異なる設定での Harris コーナー検出のパフォーマンス見積もりを示します。

表 285: cornerHarris 関数のパフォーマンス見積もりのサマリ

動作モード	動作周波数 (MHz)	設定			レイテンシ見積もり
		Sobel	ボックス	NMS 範囲	レイテンシ (ms)
1 ピクセル	300 MHz	3	3	1	7
1 ピクセル	300 MHz	3	5	1	7.1
1 ピクセル	300 MHz	3	7	1	7.1
1 ピクセル	300 MHz	5	3	1	7.2
1 ピクセル	300 MHz	5	5	1	7.2
1 ピクセル	300 MHz	5	7	1	7.2
1 ピクセル	300 MHz	7	3	1	7.22
1 ピクセル	300 MHz	7	5	1	7.22

表 285: cornerHarris 関数のパフォーマンス見積もりのサマリ (続き)

動作モード	動作周波数 (MHz)	設定			レイテンシ見積もり
		Sobel	ボックス	NMS 範囲	レイテンシ (ms)
1 ピクセル	300 MHz	7	7	1	7.22
8 ピクセル	150 MHz	3	3	1	1.7
8 ピクセル	150 MHz	3	5	1	1.7
8 ピクセル	150 MHz	3	7	1	1.7
8 ピクセル	150 MHz	5	3	1	1.71
8 ピクセル	150 MHz	5	5	1	1.71
8 ピクセル	150 MHz	5	7	1	1.71
8 ピクセル	150 MHz	7	3	1	1.8
8 ピクセル	150 MHz	7	5	1	1.8
8 ピクセル	150 MHz	7	7	1	1.8
1 ピクセル	300 MHz	3	3	2	7.1
1 ピクセル	300 MHz	3	5	2	7.1
1 ピクセル	300 MHz	3	7	2	7.1
1 ピクセル	300 MHz	5	3	2	7.21
1 ピクセル	300 MHz	5	5	2	7.21
1 ピクセル	300 MHz	5	7	2	7.21
1 ピクセル	300 MHz	7	3	2	7.22
1 ピクセル	300 MHz	7	5	2	7.22
1 ピクセル	300 MHz	7	7	2	7.22
8 ピクセル	150 MHz	3	3	2	1.8
8 ピクセル	150 MHz	3	5	2	1.8
8 ピクセル	150 MHz	3	7	2	1.8
8 ピクセル	150 MHz	5	3	2	1.81
8 ピクセル	150 MHz	5	5	2	1.81
8 ピクセル	150 MHz	5	7	2	1.81
8 ピクセル	150 MHz	7	3	2	1.9
8 ピクセル	150 MHz	7	5	2	1.91
8 ピクセル	150 MHz	7	7	2	1.92

## OpenCV との違い

xfOpenCV にはしきい値と NMS が含まれますが、OpenCV には含まれません。xfOpenCV では、すべての点が固定小数点でインプリメントされます。OpenCV では、すべてのブロックが浮動小数点でインプリメントされます。

## ヒストグラム計算 (calcHist)

calcHist 関数は、入力画像のヒストグラムを計算します。

$$H[src(x, y)] = H[src(x, y)] + 1$$

ここで、H は 256 個の要素の配列です。

## API 構文

```
template<int SRC_T, int ROWS, int COLS, int NPC=1>
void calcHist(xf::Mat<SRC_T, ROWS, COLS, NPC> & _src, uint32_t *histogram)
```

## パラメーターの説明

次の表に、テンプレートと関数のパラメーターを説明します。

表 286: calcHist 関数パラメーターの説明

パラメーター	説明
SRC_T	入力ピクセルのデータ型。8 ビット、符号なし、1 および 3 チャンネル (XF_8UC1、XF_8UC3) のみサポート。
ROWS	入力および出力画像の最大高さ。
COLS	入力および出力画像の最大幅 (8 ピクセル動作の場合は 8 の倍数で指定)
NPC	1 サイクルごとに処理されるピクセル数
_src	入力画像
histogram	256 個の要素の出力配列

## リソース使用量

次の表に、Xczu9eg-ffvb1156-1-i-es1 FPGA で Vivado HLS 2019.1 ツールを使用して生成された、通常動作 (1 ピクセル モードで 300 MHz) およびリソース最適化 (8 ピクセル モードで 150 MHz) 設定での calcHist 関数のリソース使用量を示します。

表 287: calcHist 関数のリソース使用量のサマリ

名前	リソース使用量	
	通常動作 (1 ピクセル)	リソース最適化 (8 ピクセル)
BRAM_18K	2	16
DSP48E	0	0
FF	196	274
LUT	240	912
CLB	57	231

次の表に、Xczu9eg-ffvb1156-1-i-es1 FPGA で 3 チャンネルの 4K 画像を 300 MHz で処理するために、Vivado HLS 2019.1 ツールを使用して生成された、通常動作 (1 ピクセル) の calcHist 関数のリソース使用量を示します。

表 288: calcHist 関数のリソース使用量のサマリ

名前	リソース使用量
	通常動作 (1 ピクセル)
BRAM_18K	8
DSP48E	0

表 288: calcHist 関数のリソース使用量のサマリ (続き)

名前	リソース使用量
	通常動作 (1 ピクセル)
FF	381
LUT	614
CLB	134

### パフォーマンス見積もり

次の表に、Xczu9eg-ffvb1156-1-i-es1 FPGA で Vivado HLS 2019.1 ツールを使用して生成された、通常動作 (1 ピクセル モードで 300 MHz) およびリソース最適化 (8 ピクセル モードで 150 MHz) 設定での calcHist 関数のパフォーマンス見積もりを示します。

表 289: calcHist 関数のパフォーマンス見積もりのサマリ

動作モード	レイテンシ見積もり
	最大 (ms)
1 ピクセル	6.9
8 ピクセル	1.7

## ヒストグラム均一化 (equalizeHist)

equalizeHist 関数は、入力画像またはビデオに対してヒストグラム均一化処理を実行します。画像のコントラストを改善し、強度範囲を拡張します。この関数は、1 つの分布 (ヒストグラム) を別の分布 (より幅が広く強度値がより均一に分配されている) にマップし、強度が範囲全体に拡張されるようにします。

ヒストグラム  $H[i]$  に対し、累積分布  $H'[i]$  は次のようになります。

$$H'[i] = \sum_{0 \leq j < i} H[j]$$

均一化された画像の強度は次のように計算されます。

$$dst(x, y) = H'(src(x, y))$$

### API 構文

```
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)
```

### パラメーターの説明

次の表に、テンプレートと関数のパラメーターを説明します。

表 290: equalizeHist 関数のパラメーターの説明

パラメーター	説明
SRC_T	入力および出力のピクセルのデータ型。8 ビット、符号なし、1 チャンネル (XF_8UC1) のみサポート。
ROWS	入力および出力画像の最大高さ。
COLS	入力および出力画像の最大幅 (8 ピクセル動作の場合は 8 の倍数で指定)
NPC	1 サイクルごとに処理されるピクセル数
_src	入力画像
_src1	入力画像
_dst	出力画像

### リソース使用量

次の表に、Xczu9eg-ffvb1156-1-i-es1 FPGA で Vivado HLS 2019.1 ツールを使用して生成された、通常動作 (1 ピクセル モードで 300 MHz) およびリソース最適化 (8 ピクセル モードで 150 MHz) 設定での equalizeHist 関数のリソース使用量を示します。

表 291: equalizeHist 関数のリソース使用量のサマリ

動作モード	動作周波数 (MHz)	使用量の見積もり				
		BRAM_18K	DSP_48E	FF	LUT	CLB
1 ピクセル	300	4	5	3492	1807	666
8 ピクセル	150	25	5	3526	2645	835

### パフォーマンス見積もり

次の表に、Xczu9eg-ffvb1156-1-i-es1 FPGA で Vivado HLS 2019.1 ツールを使用して生成された、通常動作 (1 ピクセル モードで 300 MHz) およびリソース最適化 (8 ピクセル モードで 150 MHz) 設定での equalizeHist 関数のパフォーマンス見積もりを示します。

表 292: equalizeHist 関数のパフォーマンス見積もりのサマリ

動作モード	レイテンシ見積もり
	最大 (ms)
1 クロックごとに 1 ピクセル動作	13.8
8 クロックごとに 1 ピクセル動作	3.4

## HOG (HOGDescriptor)

HOG (Histogram of Oriented Gradients) は、コンピューター ビジョンで物体検出のために使用される特徴ディスクリプターです。この方法で生成された特徴ディスクリプターは、歩行者の検出に広く使用されます。

この方法では、画像中の局所領域内における勾配方向の発生回数をカウントします。HOG は均一間隔のセルの高密度グリッドで計算され、精度を向上するためオーバーラップしたブロックが正規化されます。HOG は、画像内の物体の外観と形状を強度勾配またはエッジ方向の分布で記述できるという概念に基づいています。

関数には RGB とグレイ入力の両方を入力できます。RGB モードでは、各プレーンに対して勾配が個別に計算され、大きいものが選択されます。指定された設定では、ウィンドウのサイズは 64x128、ブロックのサイズは 16x16 です。

## API 構文

```
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);
```

## パラメーターの説明

次の表に、テンプレートのパラメーターを説明します。

表 293: HOGDescriptor テンプレートのパラメーターの説明

パラメーター	説明
WIN_HEIGHT	ウィンドウのピクセル行の数。画像の行数までの 8 の倍数値を指定します。
WIN_WIDTH	ウィンドウのピクセル列の数。画像の列数までの 8 の倍数値を指定します。
WIN_STRIDE	2 つの隣接するウィンドウ間のピクセル幅。8 に固定されています。
BLOCK_HEIGHT	ブロックの高さ。16 に固定されています。
BLOCK_WIDTH	ブロックの幅。16 に固定されています。
CELL_HEIGHT	セルの行数。8 に固定されています。
CELL_WIDTH	セルの列数。8 に固定されています。
NOB	セルのヒストグラム ビンの数。9 に固定されています。
DESC_SIZE	出力ディスクリプターのサイズ。
IMG_COLOR	画像のタイプ。XF_GRAY または XF_RGB に設定します。
OUTPUT_VARIANT	XF_HOG_RB または XF_HOG_NRB に設定する必要があります。
SRC_T	入力ピクセルのデータ型。グレイの場合は XF_8UC1、カラーの場合は XF_8UC4。
DST_T	出力ディスクリプターのタイプ。XF_32UC1 に設定する必要があります。
ROWS	処理される画像の行数。
COLS	処理される画像の列数。
NPC	1 サイクルごとに処理されるピクセル数。XF_NPPC1 (1 サイクルごとに 1 ピクセル動作) のみサポート。
USE_URAM	一部のストレージ構造をブロック RAM ではなく UltraRAM にマップ。

次の表に、関数のパラメーターを説明します。

表 294: HOGDescriptor 関数のパラメーターの説明

パラメーター	説明
_in_mat	xf::Mat 型の入力画像
_desc_mat	xf::Mat 型の出力ディスクリプター

説明:

- NO: 通常動作 (1 ピクセル処理)
- RB: 繰り返しブロック (ディスクリプター データはウィンドウ単位で記述される)



- NRB: 非繰り返しブロック (書き込み数を削減するため、ディスクリプター データはウィンドウ単位で記述される)。

**注記:** RB モードでは、ブロック データはオーバーラップ ウィンドウを考慮してメモリに書き込まれます。NRB モードでは、ブロック データはウィンドウ オーバーラップを考慮せずに出力ストリームに直接書き込まれます。ホスト側でオーバーラップを処理する必要があります。

## リソース使用量

次の表に、Xczu9eg-ffvb1156-1-i-es1 で解像度 1920x1080 の画像を処理するために、Vivado HLS 2019.1 ツールを使用して生成された通常動作 (1 ピクセル) での HOGDescriptor 関数のリソース使用量を示します。

表 295: HOGDescriptor 関数のリソース使用量のサマリ

リソース	1 ピクセル動作 (300 MHz) での使用量			
	NRB		RB	
	グレー	RGB	グレー	RGB
BRAM_18K	43	49	171	177
DSP48E	34	46	36	48
FF	15365	15823	15205	15663
LUT	12868	13267	13443	13848

次の表に、xczu7ev-ffvc1156-2-e で 300 MHz で解像度 1920x1080 の画像を処理するために、SDx 2019.1 ツールを使用して生成された通常動作 (1 ピクセル) モードでの HOGDescriptor 関数のリソース使用量を示します。

表 296: UltraRAM をイネーブルにした場合の HOGDescriptor 関数のリソース使用量のサマリ

リソース	1 ピクセル動作 (300 MHz) での使用量			
	NRB		RB	
	グレー	RGB	グレー	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

## パフォーマンス見積もり

次の表に、Xczu9eg-ffvb1156-1-i-es1 で解像度 1920x1080p の画像を処理するために、Vivado HLS 2019.1 ツールを使用して生成された異なる設定での HOGDescriptor() 関数のリソース使用量を示します。

表 297: HOGDescriptor 関数のパフォーマンス見積もりのサマリ

動作モード	動作周波数 (MHz)	レイテンシ見積もり	
		最小 (ms)	最大 (ms)
NRB-グレー	300	6.98	8.83
NRB-RGBA	300	6.98	8.83
RB-グレー	300	176.81	177
RB-RGBA	300	176.81	177

## OpenCV との違い

OpenCV との違いは、次のとおりです。

### 1. 境界の処理

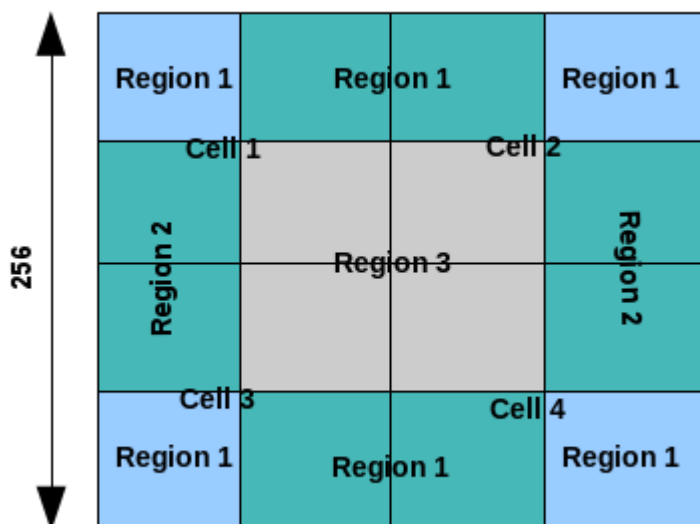
OpenCV で勾配の計算に使用される境界の処理は BORDER\_REFLECT\_101 であり、境界のパディングは近傍ピクセルの反射です。ザイリンクス インプリメンテーションでは、境界の処理に BORDER\_CONSTANT (0 パディング) が使用されます。

### 2. ガウシアン重み付け

ガウシアン重みはブロック全体でピクセルに乗算されます。ブロックには 256 ピクセルあり、ブロックの各位置が対応するガウシアン重みで乗算されます。HLS インプリメンテーションでは、ガウシアン重み付けは実行されません。

### 3. セル単位の補間

ピクセルの強度値は、対応するピンのブロック内の異なるセルに分配されます。



領域 1 のピクセルは対応するセルにのみ属し、領域 2 および 3 のピクセルはそれぞれ隣接する 2 つのセルおよび 4 つのセルに保管されます。HLS インプリメンテーションではこの操作は実行されません。

### 4. 出力の処理

OpenCV の出力は列優先形式です。HLS インプリメンテーションでは、出力は行優先形式です。また、特徴ベクターは HLS インプリメンテーションでは固定小数点型 Q0.16 ですが、OpenCV では浮動小数点です。

## 制限

1. 設定は [Dalal のインプリメンテーション](#) に制限されます。
2. 画像の高さおよび幅は、それぞれセルの高さおよび幅の倍数である必要があります。

<sup>1</sup> N. Dalal, B. Triggs 著: 『Histograms of oriented gradients for human detection』、IEEE Computer Society Conference on Computer Vision and Pattern Recognition、2005 年

## ハフ変換に基づく線分の抽出 (Houghlines)

ここでの `HoughLines` 関数は、OpenCV の `HoughLines` 標準と等価です。 `HoughLines` 関数は、バイナリ画像の直線を検出するために使用されます。ハフ変換を適用するには、エッジ検出処理を実行しておく必要があります。ハフ変換への入力、エッジ検出済みのバイナリ画像です。バイナリ画像の各点  $(x_i, y_i)$  に対して、その点を通る直線のグループは次のように定義されます。

$$\rho = x_i \cos(\theta) + y_i \sin(\theta)$$

$(\rho, \theta)$  の各ペアは、点  $(x_i, y_i)$  を通過する直線を表します。各点を通る直線グループのこれらの  $(\rho, \theta)$  ペアは、  $(\rho, \theta)$  面で正弦曲線を形成します。  $N$  個の点の正弦が  $(\rho, \theta)$  面で交差する場合、これらの交差点  $(\rho_1, \theta_1)$  は  $N$  個の点を通る直線を表します。 `HoughLines` 関数では、アキュムレータを使用して、  $(\rho, \theta)$  面の交差点のカウント (投票とも呼ばれる) を記録します。投票後、中央投票値が近傍の投票およびしきい値より大きいかどうかをチェックして中央投票値を有効または 0 にすることにより間引き、無効な直線を除去します。最後に、  $(\rho, \theta)$  の望ましい最大直線数 (`LINESMAX`) を出力として返します。

原点は画像の中心 (`Floor(COLS/2)`, `Floor(ROWS/2)`) であると想定されます。  $\rho$  および  $\theta$  の範囲は次のとおりです。

$$\theta = [0, \pi)$$

$$\rho = [-DIAG/2, DIAG/2), \text{ where } DIAG = \text{cvRound}\{\text{SquareRoot}((COLS * COLS) + (ROWS * ROWS))\}$$

使いやすさのため、入力の角度 `THETA`、`MINTHETA`、および `MAXTHETA` は度で指定し、出力  $\theta$  はラジアンで表されます。角度分解 `THETA` は整数として宣言されますが、Q6.1 フォーマットの値として扱われます。たとえば、`THETA=3` は関数で使用される分解が 1.5 度であることを示します。出力  $(\rho, \theta)$  が直線の描画に使用された際、原点が画像の中心であることに注意してください。

### API 構文

```
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)
```

### パラメーターの説明

次の表に、テンプレートと関数のパラメーターを説明します。

表 298: `HoughLines` 関数のパラメーターの説明

パラメーター	説明
RHO	アキュムレータの距離精度 (ピクセル数)。
THETA	アキュムレータの角度分解 (度および Q6.1 フォーマット)。
MAXLINES	検出する直線の最大数。
MINTHETA	直線をチェックする最小角度 (度)。
MAXTHETA	直線をチェックする最大角度 (度)。

表 298: HoughLines 関数のパラメーターの説明 (続き)

パラメーター	説明
DIAG	画像の対角。cvRound(sqrt(rows*rows + cols*cols)/RHO) である必要があります。
SRC_T	入力ピクセルのデータ型。8 ビット、符号なし、1 チャンネル (XF_8UC1) のみサポート。
ROWS	入力画像の最大高さ。
COLS	入力画像の最大幅。
NPC	1 サイクルごとに処理されるピクセル数。1 ピクセル動作 (XF_NPPC1) のみサポート。
_src_mat	入力画像は、8 ビットのシングル チャンネル バイナリ画像である必要あり。
outputrho	rho 値の出力配列。rho は座標原点 (画像の中心) からの距離です。
outputtheta	theta 値の出力配列。theta は直線の回転角度 (ラジアン) です。
threshold	アキュムレータのしきい値パラメーター。投票数がしきい値を超える直線のみが返されます。
linesmax	直線の最大数。

## リソース使用量

次の表に、Xczu9eg-ffvb1156-1-i-es1 でグレースケール HD (1080x1920) 画像で 512 線を処理するために、Vivado HLS 2019.1 ツールを使用して生成された異なる設定でのカーネルのリソース使用量を示します。

表 299: HoughLines 関数のリソース使用量のサマリ

名前	リソース使用量
	THETA=1、RHO=1
BRAM_18K	542
DSP48E	10
FF	60648
LUT	56131

## パフォーマンス見積もり

次の表に、Xczu9eg-ffvb1156-1-i-es1 でグレースケール HD (1080x1920) 画像で 512 線を処理するために、Vivado HLS 2019.1 ツールを使用して生成された異なる設定でのカーネルのパフォーマンス見積もりを示します。

表 300: HoughLines 関数のパフォーマンス見積もりのサマリ

動作モード	動作周波数 (MHz)	レイテンシ見積もり
		最大 (ms)
THETA=1、RHO=1	300	12.5

## ピラミッド アップ (pyrUp)

pyrUp 関数は、画像のアップサンプリング アルゴリズムです。まず、各入力行および列の後に 0 行と 0 列を挿入して、出力画像のサイズを作成します。出力画像のサイズは、常に  $(2*rows \times 2*columns)$  になります。この後、0 でパディングされた画像がガウシアン画像フィルターを使用して平滑化されます。ピラミッド アップ関数のガウシアン フィルターでは、次のような固定フィルター カーネルが使用されます。

$$\frac{1}{256} \begin{bmatrix} 1 & 4 & 6 & 4 & 1 \\ 4 & 16 & 24 & 16 & 4 \\ 6 & 24 & 36 & 24 & 6 \\ 4 & 16 & 24 & 16 & 4 \\ 1 & 4 & 6 & 4 & 1 \end{bmatrix}$$

ただし、0 パディングによって削減されるピクセル強度を補強するため、各出力ピクセルは 4 で乗算されます。

## API 構文

```
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)
```

## パラメーターの説明

次の表に、テンプレートと関数のパラメーターを説明します。

表 301: pyrUp 関数のパラメーターの説明

パラメーター	説明
TYPE	入力および出力のピクセルのデータ型。8 ビット、符号なし、1 および 3 チャンネル (XF_8UC1、XF_8UC3) のみサポート。
ROWS	このカーネルのハードウェアを構築するための最大高さまたは出力行数。
COLS	このカーネルのハードウェアを構築するための最大幅または出力列数。
NPC	1 サイクルごとに処理されるピクセル数。現時点では、カーネルで 1 サイクルごとに 1 ピクセル処理 (XF_NPPC1) のみをサポート。
_src	入力画像ストリーム
_dst	出力画像ストリーム

## リソース使用量

次の表に、最大入力画像サイズ 1920x1080 ピクセルの場合の各サイクル 1 ピクセルのインプリメンテーションでの pyrUp のリソース使用量を示します。これは、300 MHz のザイリンクス Xczu9eg-ffvb1156-1-i-es1 FPGA を Vivado HLS 2019.1 で合成した結果です。

表 302: pyrUp 関数のリソース使用量のサマリ

動作モード	動作周波数 (MHz)	使用量の見積もり			
		LUT	FF	DSP	BRAM
1 ピクセル	300	1124	1199	0	10

次の表に、最大入力画像サイズ 4K の BGR に対して、各サイクル 1 ピクセルのインプリメンテーションでの pyrUp のリソース使用量を示します。これは、300 MHz のザイリンクス Xczu9eg-ffvb1156-1-i-es1 FPGA を Vivado HLS 2019.1 で合成した結果です。

表 303: pyrUp 関数のリソース使用量のサマリ

動作モード	動作周波数 (MHz)	使用量の見積もり			
		LUT	FF	DSP	BRAM
1 ピクセル	300	2074	2176	0	59

### パフォーマンス見積もり

次の表に、サイリンクス Xczu9eg-ffvb1156-1-i-es1 FPGA 用に Vivado HLS 2019.1 を使用して生成された PyrUp のパフォーマンス見積もりを示します。

表 304: pyrUp 関数のパフォーマンス見積もりのサマリ

動作モード	動作周波数 (MHz)	入力画像サイズ	レイテンシ見積もり
			最大 (ms)
1 ピクセル	300	1920x1080	27.82

## ピラミッド ダウン (pyrDown)

pyrDown 関数は、画像をダウンスケールする前に平滑化する画像のダウンサンプリング アルゴリズムです。画像は、次のカーネルのガウシアン フィルターを使用して平滑化されます。

$$\frac{1}{256} \begin{bmatrix} 1 & 4 & 6 & 4 & 1 \\ 4 & 16 & 24 & 16 & 4 \\ 6 & 24 & 36 & 24 & 6 \\ 4 & 16 & 24 & 16 & 4 \\ 1 & 4 & 6 & 4 & 1 \end{bmatrix}$$

ダウンスケールは、偶数行と偶数列のピクセルを破棄することにより実行されます。結果の画像サイズは

$$\left( \frac{rows + 1}{2} \quad \frac{columns + 1}{2} \right) \text{ です。}$$

### API 構文

```
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)
```

### パラメーターの説明

次の表に、テンプレートと関数のパラメーターを説明します。

表 305: pyrDown 関数のパラメーターの説明

パラメーター	説明
TYPE	入力および出力のピクセルのデータ型。8 ビット、符号なし、1 および 3 チャンネル (XF_8UC1、XF_8UC3) のみサポート。
ROWS	このカーネルのハードウェアを構築するための最大高さまたは入力行数。
COLS	このカーネルのハードウェアを構築するための最大幅または入力列数。
NPC	1 サイクルごとに処理されるピクセル数。現時点では、カーネルで 1 サイクルごとに 1 ピクセル処理 (XF_NPPC1) のみをサポート。
USE_URAM	ストレージ構造を UltraRAM にマップ
_src	入力画像ストリーム
_dst	出力画像ストリーム

### リソース使用量

次の表に、最大入力画像サイズ 1920x1080 ピクセルの場合の 1 サイクルごとに 1 ピクセルのインプリメンテーションでの pyrDown のリソース使用量を示します。これは、300 MHz のザイリンクス Xczu9eg-ffvb1156-1-i-es1 FPGA を Vivado HLS 2019.1 で合成した結果です。

表 306: pyrDown 関数のリソース使用量のサマリ

動作モード	動作周波数 (MHz)	使用量の見積もり			
		LUT	FF	DSP	BRAM
1 ピクセル	300	1171	1238	1	5

次の表に、最大入力画像サイズ 4K の BGR 画像に対して、1 サイクルごとに 1 ピクセルのインプリメンテーションでの pyrDown のリソース使用量を示します。これは、300 MHz のザイリンクス Xczu9eg-ffvb1156-1-i-es1 FPGA を Vivado HLS 2019.1 で合成した結果です。

表 307: pyrDown 関数のリソース使用量のサマリ

動作モード	動作周波数 (MHz)	使用量の見積もり			
		LUT	FF	DSP	BRAM
1 ピクセル	300	2158	1983	2	30

次の表に、最大入力画像サイズ 3840x2160 ピクセルの場合の 1 サイクルごとに 1 ピクセルのインプリメンテーションでの pyrDown のリソース使用量を示します。これは、300 MHz のザイリンクス xczu7eg-ffvb1156-1 FPGA を SDx 2019.1 で合成した結果です。

表 308: UltraRAM をイネーブルにした場合の PyrDown 関数のリソース使用量のサマリ

動作モード	動作周波数 (MHz)	使用量の見積もり				
		LUT	FF	DSP	BRAM	URAM
1 ピクセル	300	1171	1243	0	0	1

## パフォーマンス見積もり

次の表に、ザイリンクス Xczu9eg-ffvb1156-1-i-es1 FPGA 用に Vivado HLS 2019.1 を使用して生成された PyrDown のパフォーマンス見積もりを示します。

表 309: pyrDown 関数のパフォーマンス見積もりのサマリ

動作モード	動作周波数 (MHz)	入力画像サイズ	レイテンシ見積もり
			最大 (ms)
1 ピクセル	300	1920x1080	6.99

## InitUndistortRectifyMapInverse

InitUndistortRectifyMapInverse 関数は、カメラパラメーターのセットに基づいて mapx および mapy を生成します (mapx および mapy は xf::remap 関数の入力)。つまり、デスティネーションの位置 (u, v) の各ピクセルに対して、ソース画像 (カメラからの元の画像) の該当する座標を計算します。InitUndistortRectifyMapInverse モジュールはハードウェア用に最適化されているので、回転行列の反転は合成可能なロジック外で計算されます。入力は固定小数点なので、浮動小数点カメラパラメーターは Q12.20 に変換された型にする必要があります。

## API 構文

```
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)
```

## パラメーターの説明

次の表に、テンプレートと関数のパラメーターを説明します。

表 310: InitUndistortRectifyMapInverse 関数のパラメーターの説明

パラメーター	説明
CM_SIZE	コンパイル時間 (3x3 行列の場合は 9) に設定する必要あり
DC_SIZE	コンパイル時間 (4、5、または 8) に設定する必要あり
MAP_T	出力マップのデータ型で、XF_32FC1 にする必要あり
ROWS	出力マップを生成するのに必要な最大画像高さ
COLS	出力マップを生成するのに必要な最大画像幅
NPC	1 サイクルごとのピクセル数。この関数では、1 サイクルごとに 1 ピクセルしかサポートされないの で、XF_NPPC1 に設定
cameraMatrix	古い座標システムでカメラを表す入力行列
distCoeffs	入力変形係数 (k1,k2,p1,p2[,k3[,k4,k5,k6]])
ir	入力変換行列は Invert(newCameraMatrix*R) と同じで、newCameraMatrix が新しい座標システムの カメラを示し、R は回転行列です。この処理は、合成可能なブロック外で実行されます。
_mapx_mat	mapx を含む mat オブジェクトを出力。
_mapy_mat	mapy を含む mat オブジェクトを出力。
_cm_size	3x3 行列の場合 9



表 310: InitUndistortRectifyMapInverse 関数のパラメーターの説明 (続き)

パラメーター	説明
_dc_size	4、5、または 8。0 の場合、変形なし

## inRange

inRange 関数は、画像 src のピクセルが指定の範囲内にあるかどうかをチェックします。src(x,y) が指定のしきい値の範囲内である場合は dst(x,y) は 255 に設定され、範囲外の場合は 0 に設定されます。

$$\text{Dst}(l) = \text{lowerb} \leq \text{src}(l) \leq \text{upperb}$$

(x,y) はピクセルの空間座標です。

### API 構文

```
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)
```

### パラメーターの説明

次の表に、テンプレートと関数のパラメーターを説明します。

表 311: inRange 関数のパラメーターの説明

パラメーター	説明
SRC_T	入力ピクセルのデータ型。8 ビット、符号なし、1 および 3 チャンネル (XF_8UC1、XF_8UC3) をサポート。
ROWS	入力および出力画像の最大高さ。
COLS	入力および出力画像の最大幅。N ピクセルの並列処理の場合、幅は N の倍数で指定。
NPC	1 サイクルごとに処理されるピクセル数。1 ピクセルの場合は XF_NPPC1、8 ピクセルの場合は XF_NPPC8。
src	入力画像
dst	出力画像
lower_thresh	しきい値の下限
upper_thresh	しきい値の上限

### リソース使用量

次の表に、Xczu9eg-ffvb1156-1-i-es1 FPGA で Vivado HLS 2019.1 バージョン ツールを使用して生成された、リソース最適化 (8 ピクセル) モードおよび通常モードにおける InRange 関数のリソース使用量を示します。

表 312: inRange 関数のリソース使用量のサマリ

名前	リソース使用量	
	1 クロックごとに 1 ピクセル動作	8 クロックごとに 1 ピクセル動作
	300 MHz	150 MHz
BRAM_18K	0	0
DSP48E	0	0
FF	86	154
LUT	60	148
CLB	15	37

### パフォーマンス見積もり

次の表に、Xczu9eg-ffvb1156-1-i-es1 FPGA でグレースケール HD (1080x1920) 画像を処理するために、Vivado HLS 2019.1 ツールを使用して生成された異なる設定でのカーネルのパフォーマンス見積もりを示します。

表 313: inRange 関数のパフォーマンス見積もりのサマリ

動作モード	レイテンシ見積もり	
	動作周波数 (MHz)	レイテンシ (ms)
1 ピクセル	300	6.9
8 ピクセル	150	1.7

## 積分画像 (integral)

`integral` 関数では、入力の積分画像が計算されます。各出力ピクセルは、そのピクセルの上および左にあるすべてのセルの合計です。

$$dst(x, y) = sum(x, y) = sum(x, y) + sum(x - 1, y) + sum(x, y - 1) - sum(x - 1, y - 1)$$

### API 構文

```
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)
```

### パラメーターの説明

次の表に、テンプレートと関数のパラメーターを説明します。

表 314: integral 関数のパラメーターの説明

パラメーター	説明
SRC_TYPE	入力ピクセルのデータ型。8 ビット、符号なし、1 チャンネル (XF_8UC1) のみサポート。
DST_TYPE	出力ピクセルのデータ型。32 ビット、符号なし、1 チャンネル (XF_32UC1) のみサポート。
ROWS	入力および出力画像の最大高さ。
COLS	入力および出力画像の最大幅。

表 314: integral 関数のパラメーターの説明 (続き)

パラメーター	説明
NPC	1 サイクルごとに処理されるピクセル数。XF_NPPC1 (1 サイクルごとに 1 ピクセル動作) のみサポート。
_src_mat	入力画像
_dst_mat	出力画像

## リソース使用量

次の表に、ザイリンクス Xczu9eg-ffvb1156-1-i-es1 FPGA でグレースケール HD (1080x1920) 画像を処理するために、Vivado HLS 2019.1 ツールを使用して生成された異なる設定でのカーネルのリソース使用量を示します。

表 315: integral 関数のリソース使用量のサマリ

名前	リソース使用量
	1 ピクセル
	300 MHz
BRAM_18K	4
DSP48E	0
FF	613
LUT	378
CLB	102

## パフォーマンス見積もり

次の表に、ザイリンクス Xczu9eg-ffvb1156-1-i-es1 でグレースケール HD (1080x1920) 画像を処理するために、Vivado HLS 2019.1 ツールを使用して生成された異なる設定でのカーネルのパフォーマンス見積もりを示します。

表 316: integral 関数のパフォーマンス見積もりのサマリ

動作モード	レイテンシ見積もり	
	動作周波数 (MHz)	レイテンシ (ms)
1 ピクセル	300	7.2

## 高密度ピラミッド型 LK オプティカル フロー (DensePyrOpticalFlow)

オプティカル フローは、物体またはカメラの動きのため発生する 2 つの連続するフレーム間における画像物体の動きのパターンです。これは 2D ベクター フィールドであり、各ベクターは 1 つ目のフレームから 2 つ目のフレームの点の移動を表す変位ベクターです。

オプティカル フローは、次を前提として実行されます。

- 物体のピクセル強度は、連続するフレーム間でそれほど変動しない
- 近傍ピクセルも同じように動く

最初のフレームにピクセル  $I(x, y, t)$  があるとします。3 つ目の次元として時間  $(t)$  が追加されています。画像のみを処理する場合は、時間は必要ありません。時間  $dt$  後の次のフレームで、ピクセルが距離  $(dx, dy)$  だけ移動します。これらのピクセルは同じであり強度は変化しないので、次のことが言えます。

$$I(x, y, t) = I(x + dx, y + dy, t + dt)$$

右辺のテイラー級数近似を求め、共通項を取り除き、 $dt$  で割ると、次の式が得られます。

$$f_x u + f_y v + f_t = 0$$

$$f_x = \frac{\delta f}{\delta x}, \quad f_y = \frac{\delta f}{\delta y}, \quad u = \frac{dx}{dt}, \quad \text{および} \quad v = \frac{dy}{dt} \quad \text{です。}$$

上記の式をオプティカルフローと呼びます。 $f_x$  および  $f_y$  は画像勾配で、 $f_t$  は時間の経過に伴う勾配です。ただし、 $(u, v)$  は不明です。これらの不明な変数のためこの方程式を解くことはできないので、この問題を解くために複数の方法が提供されています。1 つの方法は Lucas-Kanade (LK) です。先ほど近傍のすべてのピクセルが同じように動く想定しました。Lucas-Kanade 法では、点を囲む WINDOW\_SIZE テンプレートパラメーターで指定されたサイズのパッチが使用されます。つまり、パッチ内のすべての点が同じように動く想定されます。これらの点に対しては、 $(f_x, f_y, f_t)$  を求めることが可能です。これで、問題は 2 つの不明な変数を含む WINDOW\_SIZE \* WINDOW\_SIZE 方程式を解くことになり、これは重複決定されます。より良い方法は、最小二乗適合法を使用する方法です。次に、2 つの方程式と 2 つの不明な変数を含む問題の最終的な解を示します。

$$\begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} \sum f_{x_i}^2 & \sum f_{x_i} f_{y_i} \\ \sum f_{x_i} f_{y_i} & \sum f_{y_i}^2 \end{bmatrix}^{-1} \begin{bmatrix} -\sum f_{x_i} f_{t_i} \\ -\sum f_{y_i} f_{t_i} \end{bmatrix}$$

このソリューションは、大きな動きがあるためにピラミッドが使用される場合はエラーとなります。ピラミッドを上に登ると小さな動きが削除され、大きな動きが小さな動きとなりますので、Lucas-Kanade 法を適用し、そのスケールでのオプティカルフローを取得します。

## API 構文

```
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)
```

## パラメーターの説明

次の表に、テンプレートと関数のパラメーターを説明します。

表 317: densePyrOpticalFlow 関数のパラメーターの説明

パラメーター	説明
NUM_PYR_LEVELS	オプティカルフローの計算に使用する画像のピラミッドレベル数。
NUM_LINES	一時的な勾配を見つけるために使用されるリマップアルゴリズム用に格納する行数。
WINSIZE	オプティカルフローを計算するウィンドウサイズ。

表 317: densePyrOpticalFlow 関数のパラメーターの説明 (続き)

パラメーター	説明
FLOW_WIDTH、 FLOW_INT	符号付きフロー ベクター データ型を定義するデータ幅と整数ビット数。整数ビットには、符号付きビットが含まれます。 デフォルトのデータ型は、10 個の整数ビットと 6 個の小数ビットを持つ 16 ビット符号付きワードです。
TYPE	入力画像のピクセルのデータ型。XF_8UC1 のみをサポート。
ROWS	このカーネルのハードウェアを構築するための最大高さまたは行数。
COLS	このカーネルのハードウェアを構築するための最大幅または列数。
NPC	カーネルで 1 クロック サイクルごとに処理する必要のあるピクセル数。1 サイクルごとに 1 ピクセル (XF_NPPC1) のみをサポート。
USE_URAM	一部のストレージ構造を UltraRAM にマップ
_curr_img	1 つ目の入力画像ストリーム
_next_img	1 つ目の画像に対してオプティカル フローを計算する 2 つ目の入力画像
_streamFlowin	オプティカル フロー用の 32 ビットにパックされた U および V フロー ベクター入力。31 ~ 16 のビットはフロー ベクター U を表し、15 ~ 0 のビットはフロー ベクター V を表します。
_streamFlowout	オプティカル フローの計算後に 32 ビットにパックされた U および V フロー ベクター出力。31 ~ 16 のビットはフロー ベクター U を表し、15 ~ 0 のビットはフロー ベクター V を表します。
level	アルゴリズムが現在オプティカル フローを計算している画像のピラミッド レベル。
scale_up_flag	フロー ベクターのスケール アップをイネーブルにするフラグ。1 つの画像ピラミッド レベルから別のピラミッド レベルに切り替える際にホストで設定されます。
scale_in	フロー ベクターをスケール アップするための浮動小数点スケール アップ係数。 この値は $(\text{previous\_rows}-1)/(\text{current\_rows}-1)$ です。1 つの画像ピラミッド レベルから別のピラミッド レベルに切り替える際は、1 ではない値になります。
init_flag	最上位ピラミッド レベルの最初の反復のフロー ベクターを 0 に初期化するフラグ。このフラグは、最上位ピラミッド レベルの最初の反復 (ピラミッドの最小の画像) で設定する必要があります。その他の反復では、このフラグをオフにする必要があります。

## リソース使用量

次の表に、1920x1080 ピクセルの画像サイズに対してウィンドウ サイズ 11 でオプティカル フローを計算する場合の、各サイクル 1 ピクセルのインプリメンテーションにおける densePyrOpticalFlow のリソース使用量を示します。これは、300 MHz のザイリンクス xczu9eg-ffvb1156-2L-e FPGA を Vivado HLS 2019.1 でインプリメンテーションした結果です。

表 318: densePyrOpticalFlow 関数のリソース使用量のサマリ

動作モード	動作周波数 (MHz)	使用量の見積もり			
		LUT	FF	DSP	BRAM
1 ピクセル	300	32231	16596	52	215

## UltraRAM をイネーブルにした場合のリソース使用量

次の表に、3840X2160 ピクセルの画像サイズに対してウィンドウ サイズ 11 でオプティカル フローを計算する場合の、各サイクル 1 ピクセルのインプリメンテーションにおける densePyrOpticalFlow のリソース使用量を示します。これは、300 MHz のザイリンクス xczu7ev-ffvc1156-2 FPGA を SDx 2019.1 でインプリメンテーションした結果です。

表 319: densePyrOpticalFlow 関数のリソース使用量のサマリ

動作モード	動作周波数 (MHz)	使用量の見積もり				
		LUT	FF	DSP	BRAM	URAM
1 ピクセル	300	31164	42320	81	34	23

### パフォーマンス見積もり

次の表に、各レベルを係数 2 でスケールダウンする反復を 5 つのピラミッドレベルに対して 5 回実行した場合の、ハードウェアでの densePyrOpticalFlow 関数のパフォーマンス見積もりを示します。これは、zcu102 評価ボードでテストされています。

表 320: densePyrOpticalFlow 関数のパフォーマンス見積もりのサマリ

動作モード	動作周波数 (MHz)	画像サイズ	レイテンシ見積もり
			最大 (ms)
1 ピクセル	300	1920x1080	49.7
1 ピクセル	300	1280x720	22.9
1 ピクセル	300	1226x370	12.02

## 高密度非ピラミッド型 LK オプティカル フロー (DenseNonPyrLKOpticalFlow)

オプティカル フローは、物体またはカメラの動きのため発生する 2 つの連続するフレーム間における画像物体の動きのパターンです。これは 2D ベクター フィールドであり、各ベクターは 1 つ目のフレームから 2 つ目のフレームの点の移動を表す変位ベクターです。

オプティカル フローは、次を前提として実行されます。

- 物体のピクセル強度は、連続するフレーム間でそれほど変動しない
- 近傍ピクセルも同じように動く

最初のフレームにピクセル  $I(x, y, t)$  があるとします。3 つ目の次元として時間  $(t)$  が追加されています。画像のみを処理する場合は、時間は必要ありません。時間  $dt$  後の次のフレームで、ピクセルが距離  $(dx, dy)$  だけ移動します。これらのピクセルは同じであり強度は変化しないので、次のことが言えます。

$$I(x, y, t) = I(x + dx, y + dy, t + dt)$$

右辺のテイラー級数近似を求め、共通項を取り除き、 $dt$  で割ると、次の式が得られます。

$$f_x u + f_y v + f_t = 0$$

$$f_x = \frac{\delta f}{\delta x}, \quad f_y = \frac{\delta f}{\delta y}, \quad u = \frac{dx}{dt}, \quad \text{および} \quad v = \frac{dy}{dt} \text{ です。}$$

上記の式をオプティカルフローと呼びます。 $f_x$  および  $f_y$  は画像勾配で、 $f_t$  は時間の経過に伴う勾配です。ただし、 $(u, v)$  は不明です。これらの不明な変数のためこの方程式を解くことはできないので、この問題を解くために複数の方法が提供されています。1つの方法は Lucas-Kanade (LK) です。先ほど近傍のすべてのピクセルが同じように動くことと想定しました。Lucas-Kanade 法では、点を囲む WINDOW\_SIZE テンプレート パラメーターで指定されたサイズのパッチが使用されます。つまり、パッチ内のすべての点が同じように動くことと想定されます。これらの点に対しては、 $(f_x, f_y, f_t)$  を求めることが可能です。これで、問題は 2つの不明な変数を含む WINDOW\_SIZE \* WINDOW\_SIZE 方程式を解くことになり、これは重複決定されます。より良い方法は、最小二乗適合法を使用する方法です。次に、2つの方程式と 2つの不明な変数を含む問題の最終的な解を示します。

$$\begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} \sum f_{x_i}^2 & \sum f_{x_i} f_{y_i} \\ \sum f_{x_i} f_{y_i} & \sum f_{y_i}^2 \end{bmatrix}^{-1} \begin{bmatrix} -\sum f_{x_i} f_{t_i} \\ -\sum f_{y_i} f_{t_i} \end{bmatrix}$$

## API 構文

```
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)
```

## パラメーターの説明

次の表に、テンプレートと関数のパラメーターを説明します。

表 321: DenseNonPyrLKOpticalFlow 関数のパラメーターの説明

パラメーター	説明
タイプ	ピクセルのデータ型。サポートされるピクセル値は符号なし 8 ビット (XF_8UC1)。
ROWS	入力画像の最大行数。ハードウェア カーネルはこの行数用に構築する必要があります。
COLS	入力画像の最大列数。ハードウェア カーネルはこの列数用に構築する必要があります。
NPC	1 サイクルごとに処理されるピクセル数。XF_NPPC1 (=1) および XF_NPPC2 (=2) をサポート。
WINDOW_SIZE	オプティカル フローを計算するウィンドウ サイズ。奇数の正の整数を指定可能。
USE_URAM	ストレージ構造を UltraRAM にマップ。
frame0	1 つ目の入力画像。
frame1	2 つ目の入力画像。オプティカル フローは、frame0 と frame1 の間で計算されます。
flowx	フロー ベクターの水平要素。フロー ベクターのフォーマットは XF_32FC1 (単精度)。
flowy	フロー ベクターの垂直要素。フロー ベクターのフォーマットは XF_32FC1 (単精度)。

## リソース使用量

次の表に、300 MHz のザイリンクス Xczu9eg-ffvb1156-1-i-es1 FPGA 用に Vivado HLS 2019.1 ツールを使用して生成された 4K 画像の DenseNonPyrLKOpticalFlow のリソース使用量を示します。

表 322: DenseNonPyrLKOpticalFlow 関数のリソース使用量のサマリ

動作モード	動作周波数 (MHz)	使用量の見積もり			
		BRAM_18K	DSP_48E	FF	LUT
1 ピクセル	300	178	42	11984	7730
2 ピクセル	300	258	82	22747	15126

次の表に、300 MHz のザイリンクス Xczu7eg-ffvb1156-1 FPGA 用に SDx ツールを使用して UltraRAM をイネーブルにして生成された 4K 画像の DenseNonPyrLKOpticalFlow のリソース使用量を示します。

表 323: UltraRAM をイネーブルにした場合の DenseNonPyrLKOpticalFlow 関数のリソース使用量のサマリ

動作モード	動作周波数 (MHz)	使用量の見積もり				
		BRAM_18K	URAM	DSP_48E	FF	LUT
1 ピクセル	300	0	12	42	11803	7469
2 ピクセル	300	0	23	80	22124	13800

### パフォーマンス見積もり

次の表に、ザイリンクス Xczu9eg-ffvb1156-1-i-es1 FPGA 用に Vivado HLS 2019.1 ツールを使用して生成された、4K 画像の DenseNonPyrLKOpticalFlow 関数のパフォーマンス見積もりを示します。

表 324: DenseNonPyrLKOpticalFlow 関数のパフォーマンス見積もりのサマリ

動作モード	動作周波数 (MHz)	レイテンシ見積もり
		最大 (ms)
1 ピクセル	300	28.01
2 ピクセル	300	14.01

## カルマン フィルター

典型的なカルマン フィルターは、線形システム用に提唱されたものです。線形システムの状態空間は、次のように記述されます。

$$x_{k+1} = A_k x_k + B_k u_k + \Gamma_k \xi_k$$

$$y_k = H_k x_k + \eta_k$$

ここで、 $x_k$  は  $k^{\text{th}}$  時刻での状態ベクター、定数 (既知)  $A_k$  は  $n \times n$  状態遷移行列、定数 (既知)  $B_k$  は  $n \times m$  制御入力行列、定数 (既知)  $\Gamma_k$  は  $n \times p$  システム ノイズ入力行列、定数 (既知)  $H_k$  は  $q \times n$  計測行列、定数 (既知)  $\{u_k\}$  ( $1 \leq m, p, q \leq n$ ) は決定

論的入力シーケンスと呼ばれる  $m$  ベクターの (既知の) シーケンス、 $\{\xi_k\}$  および  $\{\eta_k\}$  はそれぞれ平均、偏差、共分散などを含む (未知の) システムおよび観測ノイズ シーケンスです。

カルマン フィルターでは、次が想定されます。



1.  $\{\xi_k\}$  および  $\{\eta_k\}$  は、ゼロ平均ガウシアン (または通常の) ホワイト ノイズのシーケンスです。つまり  $E(\eta_k) = 0$ ,  $E(\xi_k \xi_l^T) = Q_k \delta_{kl}$  および  $E(\eta_k \eta_l^T) = R_k \delta_{kl}$  で、 $\delta_{kl}$  はクロネッカーのデルタ関数、 $Q_k$  および  $R_k$  は正定値行列、 $E(u)$  はランダム変数  $u$  の期待値です。

2.  $E(\xi_k \eta_l^T) = 0 \forall k, l$

3. 初期状態  $x_0$  は  $\{\xi_k\}$  および  $\{\eta_k\}$  からは独立しており、 $E(x_0 \eta_k^T) = E(x_0 \xi_l^T) = 0 \forall k, l$  です。

$\hat{x}_{k|j} = \hat{x}_{k,j}$  は、時刻  $j$  までに計測されたデータをすべて使用した時刻  $k$  での  $x$  の見積もりです。カルマン フィルター アルゴリズムは、次の式に示すようにまとめられます。

初期化

$$\begin{cases} P_{0,0} = \text{Var}(x_0) \\ \hat{x}_{0|0} = E(x_0) \end{cases}$$

時間アップデート/予測

$$\begin{cases} \hat{x}_{k|k-1} = A_{k-1} \hat{x}_{k-1|k-1} + B_{k-1} u_{k-1} \\ P_{k,k-1} = A_{k-1} P_{k-1,k-1} A_{k-1}^T + \Gamma_{k-1} Q_{k-1} \Gamma_{k-1}^T \end{cases}$$

計測アップデート/補正

$$\begin{cases} G_k = P_{k,k-1} H_k^T (R_k + H_k P_{k,k-1} H_k^T)^{-1} \\ \hat{x}_{k|k} = \hat{x}_{k|k-1} + G_k (v_k - D_k u_k - H_k \hat{x}_{k|k-1}) \\ P_{k,k} = (I - G_k H_k) P_{k,k-1} \end{cases}$$

$P$  は見積もりエラー共分散  $n \times n$  行列、 $G_k$  はカルマンゲイン  $n \times q$  行列、 $k=1, 2, \dots$  は時刻です。

### 計算ストラテジ

カルマン フィルターの共分散計測アップデートの数値的精度は、2つの正定値配列を区別するので、インプリメンテーションで懸念事項となります。計算に有限精度を使用する場合は、これが問題となる可能性があります。このデザインでは、数値的精度/安定性の問題に対処するため、 $P$  の UDU 因数分解を使用します。

$$P_{k,k} = (I - G_k H_k) P_{k,k-1} = P_{k,k-1} - P_{k,k-1} H_k^T (R_k + H_k P_{k,k-1} H_k^T)^{-1} P_{k,k-1}$$

初期化中 (最初の反復の前)、エラー共分散行列  $P$  ( $U0\_mat$  および  $D0\_mat$ ) と、システム ノイズ共分散行列  $Q$  ( $Uq\_mat$  および  $Dq\_mat$ ) を供給する必要があります。  $U$  および  $D$  行列は、後退代入コレスキー分解を使用して取得できます。

次に、  $P$  を、  $P=UDUT$  となるように単位上三角行列  $U$  と対角行列  $D$  に後退代入コレスキー分解する方法を示します。

$U$  および  $D$  の  $n$  番目の列は、次のようになります。

$$D_{nn} = P_{nn}$$

$$U_{in} = \begin{cases} 1, & i = n \\ P_{in} / D_{nn}, & i = n-1, n-2, \dots, 1 \end{cases}$$

残りの行 ( $j=n-1, n-2, \dots, 1$ ) は、次のようになります。

$$D_{jj} = P_{jj} - \sum_{k=j+1}^n D_{kk} U_{jk}^2$$

$$U_{ij} = \begin{cases} 0, & i > j \\ 1, & i = j \\ \frac{P_{ij} - \sum_{k=j+1}^n D_{kk} U_{ik} U_{jk}}{D_{jj}}, & i = j-1, j-2, \dots, 1 \end{cases}$$

### カルマン フィルターの例

```
//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 構文

```
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)
```

## パラメーターの説明

表 325: カルマン フィルター関数のパラメーターの説明

パラメーター	使用 (✓)/不使用 (X)			説明
	初期化	時間アップデート	計測アップデート	
N_STATE	✓	✓	✓	状態変数の数 (1 ~ 128)
M_MEAS	✓	✓	✓	計測変数の数 (1 ~ 128)。M_MEAS は N_STATE 以下である必要があります。拡張カルマン フィルター (EKF) の場合は、M_MEAS は 1 である必要があります。
C_CTRL	✓	✓	✓	制御変数の数 (0 ~ 128)。C_CTRL は N_STATE 以下である必要があります。EKF の場合は、C_CTRL は 1 である必要があります。
MTU	✓	✓	✓	時間アップデートで使用する乗数の数 (1 ~ 128)。MTU は N_STATE 以下である必要があります。
MMU	✓	✓	✓	計測アップデートで使用する乗数の数 (1 ~ 128)。MMU は N_STATE 以下である必要があります。
USE_URAM	✓	✓	✓	URAM イネーブル。有効な値は 0 および 1 です。
EKF_EN	✓	✓	✓	拡張カルマン フィルター イネーブル。有効な値は 0 および 1 です。
TYPE	✓	✓	✓	入力ピクセルのデータ型。現在のところ、XF_32FC1 のみがサポートされています。
NPC	✓	✓	✓	1 サイクルごとに処理されるピクセル数。可能なオプションは XF_NPPC1 です (この関数では関係なし)。
A_mat	✓	X	X	遷移行列 A。EKF の場合は、ヤコビ行列 F は A_mat にマップされます。

表 325: カルマン フィルター関数のパラメーターの説明 (続き)

パラメーター	使用 (✓)/不使用 (X)			説明
	初期化	時間アップデート	計測アップデート	
B_mat	✓	X	X	制御行列 B。KF の場合、C_CTRL=0 のときに B_mat 引数は必要ありません。EKF の場合は、サイズ (N_STATE x 1) のダミー行列が B_mat にマップされます。
Uq_mat	✓	X	X	プロセス ノイズ共分散行列 Q の U 行列
Dq_mat	✓	X	X	プロセス ノイズ共分散行列 Q の D 行列 (対角要素のみ)
H_mat	✓	X	X	計測行列 H。EKF の場合は、ヤコビ行列 H は H_mat にマップされます。
X0_mat	✓	X	X	初期状態行列。EKF の場合は、状態遷移行列 f は X0_mat にマップされます。
U0_mat	✓	X	X	初期エラー見積もり共分散行列 P の U 行列
D0_mat	✓	X	X	初期エラー見積もり共分散行列 P の D 行列 (対角要素のみ)
R_mat	✓	X	X	計測ノイズ共分散行列 R (対角要素のみ)。EKF の場合は、M_MEAS=1 なので R の 1 つの値のみを入力します。
u_mat	X	✓	X	制御入力ベクター。KF の場合、C_CTRL=0 のときは u_mat 引数は必要ありません。EKF の場合は、監視行列 h は u_mat にマップされます。
y_mat	X	X	✓	計測ベクター。EKF の場合は、M_MEAS=1 なので、1 つの計測のみを入力します。
Xout_mat	X	✓	✓	出力状態行列
Uout_mat	X	✓	✓	出力エラー見積もり共分散行列 P の U 行列
Dout_mat	X	✓	✓	出力エラー見積もり共分散行列 P の D 行列 (対角要素のみ)
フラグ	✓	✓	✓	制御フラグレジスタ
初期化されたすべての行列 (Q および P) と同等の U、D は、U-D 方程式を使用して求められます。				

表 326: 制御フラグレジスタ

フラグ ビット	説明
0	初期化イネーブル
1	時間アップデート イネーブル
2	計測アップデート イネーブル
3	時間アップデートの $X_{out}$ イネーブル
4	時間アップデートの $U_{out}/D_{out}$ イネーブル
5	計測アップデートの $X_{out}$ イネーブル
6	計測アップデートの $U_{out}/D_{out}$ イネーブル
7	拡張カルマン フィルター用の読み出し最適化 (Uq_mat、Dq_mat、U0_mat、D0_mat、および R_mat)

## リソース使用量

次の表に、サイリンクス Xczu9eg-ffvb1156-1 FPGA 用に SDx 2019.1 ツールを使用して生成された異なる設定でのカーネルのリソース使用量を示します。

表 327: カルマン フィルター関数のリソース使用量のサマリ

名前	リソース使用量		
	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

次の表に、サイリンクス xczu7ev-ffvc1156-2-e FPGA 用に SDx 2019.1 ツールを使用して生成された USE\_URAM をイネーブルにした場合の異なる設定でのカーネルのリソース使用量を示します。

表 328: UltraRAM をイネーブルにした場合のリソース使用量

リソース	リソース使用量 (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

## パフォーマンス見積もり

次の表に、サイリンクス Xczu9eg-ffvb1156-1 FPGA 用に SDx 2019.1 ツールを使用して生成された異なる設定でのカーネルのパフォーマンスを示します (1 イテレーション)。レイテンシ見積もりは、100 回の反復の平均です。

表 329: カルマン フィルター関数のパフォーマンス見積もりのサマリ

動作モード	レイテンシ見積もり	
	動作周波数 (MHz)	レイテンシ (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

次の表に、サイリンクス xczu7ev-ffvc1156-2-e FPGA 用に SDx 2019.1 ツールを使用して生成された UltraRAM をイネーブルにした場合のカーネルのパフォーマンスを示します (1 イテレーション)。レイテンシ見積もりは、100 回の反復の平均です。

表 330: UltraRAM を使用した場合のパフォーマンス見積もり

動作モード	動作周波数 (MHz)	レイテンシ (ms)
N_STATE=64、C_CTRL=64、 M_MEAS=12、MTU=4、MMU=4	300	0.25

## 拡張カルマン フィルター

カルマン フィルターは、線形モデルの状態ベクトルを見積もります。モデルが非線形の場合は、フィルター式を取得するため線形化が実行されます。このようにして取得されたカルマン フィルターを拡張カルマン フィルターと呼びます。非線形システムの状態/空間記述には、次の形式の非線形モデルが含まれます。

$$x_{k+1} = (x_k) + T_k(x_k)\xi_k$$

$$z_k = h_k(x_k) + \eta_k$$

$f_k$  および  $h_k$  は、それぞれ  $R^n$  および  $R^q$  範囲の値関数です ( $1 \leq q \leq n$ )。行列値関数  $T_k$  は  $R^n \times R^q$  範囲であり、各  $k$  に対して、すべての  $x_k$  のコンポーネントにおける  $f_k(x_k)$  および  $h_k(x_k)$  の一階偏微分が連続したものになる必要があります。

ゼロ平均ガウシアン ホワイト ノイズ シーケンス  $\{\xi_k\}$  および  $\{\eta_k\}$  は、それぞれ  $R^p$  および  $R^q$  の範囲です ( $1 \leq p, q \leq n$ )。

リアルタイム線形化は、次の式に示すように実行されます。線形モデルの線で、初期見積もりおよび予測位置は次のようになります。

$$\hat{x}_0 = E(x_0), \quad \hat{x}_{1|0} = f_0(\hat{x}_0)$$

そして、連続する  $k=1,2,\dots$  の  $\hat{x}_k = \hat{x}_{k|k}$  に対して予測位置が使用されます。

$$\hat{x}_{k|k-1} = f_{k-1}(\hat{x}_{k-1})$$

注記:

1.  $f_k(x_k) = \begin{bmatrix} f_k^1(x_k) \\ \vdots \\ f_k^n(x_k) \end{bmatrix}$ 、ここで  $x_k = \begin{bmatrix} x_k^1 \\ \vdots \\ x_k^n \end{bmatrix}$ 、 $k$  は時間インデックス、上付き文字は行インデックス、

$$\left[ \frac{\partial f_k(x_k)}{\partial x_k} \right] = \begin{bmatrix} \frac{\partial f_k^1(x_k)}{\partial x_k^1} & \cdots & \frac{\partial f_k^1(x_k)}{\partial x_k^n} \\ \vdots & \cdots & \vdots \\ \frac{\partial f_k^n(x_k)}{\partial x_k^1} & \cdots & \frac{\partial f_k^n(x_k)}{\partial x_k^n} \end{bmatrix}$$

2.  $R^m$  は列ベクターの空間  $\mathbf{x} = [x_1 \dots x_m]^T$

時間アップデート計算の式は、次のようになります。

$$P_{k|k-1} = \left[ \frac{\partial f_{k-1}(\hat{x}_{k-1})}{\partial x_{k-1}} \right] P_{k-1|k-1} \left[ \frac{\partial f_{k-1}(\hat{x}_{k-1})}{\partial x_{k-1}} \right]^T + T_{k-1}(\hat{x}_{k-1}) Q_{k-1} T_{k-1}(\hat{x}_{k-1})^T$$

$$= F_{k-1} P_{k-1|k-1} F_{k-1}^T + T_{k-1}(\hat{x}_{k-1}) Q_{k-1} T_{k-1}(\hat{x}_{k-1})^T$$

計測アップデート計算の式は、次のようになります。

$$G_k = P_{k|k-1} \left[ \frac{\partial h_k(\hat{x}_{k|k-1})}{\partial x_k} \right]^T \left( R_k + \left[ \frac{\partial h_k(\hat{x}_{k|k-1})}{\partial x_k} \right] P_{k|k-1} \left[ \frac{\partial h_k(\hat{x}_{k|k-1})}{\partial x_k} \right]^T \right)^{-1}$$

$$\hat{x}_{k|k} = \hat{x}_{k|k-1} + G_k (v_k - h_k(\hat{x}_{k|k-1}))$$

$$P_{k|k} = \left( I - G_k \left[ \frac{\partial h_k(\hat{x}_{k|k-1})}{\partial x_k} \right] \right) P_{k|k-1} = (I - G_k H_k) P_{k|k-1}$$

#### 拡張カルマン フィルターの例

```
//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);
}
```

## 平均および標準偏差 (meanStdDev)

meanStdDev 関数では、入力画像の平均偏差と標準偏差が計算されます。出力される平均値のフォーマットは固定小数点の Q8.8 で、標準偏差値のフォーマットは Q8.8 です。平均および標準偏差は次のように計算されます。

$$\mu = \frac{\sum_{y=0}^{height} \sum_{x=0}^{width} src(x, y)}{(width * height)}$$

$$\sigma = \sqrt{\frac{\sum_{y=0}^{height} \sum_{x=0}^{width} (\mu - src(x, y))^2}{(width * height)}}$$

### API 構文

```
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)
```

### パラメーターの説明

次の表に、テンプレートと関数のパラメーターを説明します。

表 331: meanStdDev 関数のパラメーターの説明

パラメーター	説明
SRC_T	入力および出力のピクセルのデータ型。8 ビット、符号なし、1 および 3 チャンネル (XF_8UC1、XF_8UC3) のみサポート。
ROWS	処理される画像の行数。
COLS	処理される画像の列数。8 ピクセル動作の場合は 8 の倍数で指定。
NPC	1 サイクルごとに処理されるピクセル数。1 ピクセルの場合は XF_NPPC1、8 ピクセルの場合は XF_NPPC8。
_src	入力画像
_mean	画像の平均値の計算結果を示す 16 ビットのデータ ポインター。
_stddev	画像の標準偏差の計算結果を示す 16 ビットのデータ ポインター。



## リソース使用量

次の表に、Xczu9eg-ffvb1156-1-i-es1 FPGA でグレースケール HD (1080x1920) 画像を処理するために、Vivado HLS 2019.1 ツールを使用して生成された meanStdDev 関数のリソース使用量を示します。

表 332: meanStdDev 関数のリソース使用量のサマリ

動作モード	動作周波数 (MHz)	使用量の見積もり				
		BRAM_18K	DSP_48E	FF	LUT	CLB
1 ピクセル	300	0	6	896	461	121
8 ピクセル	150	0	13	1180	985	208

次の表に、Xczu9eg-ffvb1156-1-i-es1 FPGA で 3 チャンネルの 4K 画像を処理するために、Vivado HLS 2019.1 ツールを使用して生成された meanStdDev 関数のリソース使用量を示します。

表 333: meanStdDev 関数のリソース使用量のサマリ

動作モード	動作周波数 (MHz)	使用量の見積もり				
		BRAM_18K	DSP_48E	FF	LUT	CLB
1 ピクセル	300	0	7	5075	3324	725

## パフォーマンス見積もり

次の表に、Xczu9eg-ffvb1156-1-i-es1 でグレースケール HD (1080x1920) 画像を処理するために、Vivado HLS 2019.1 ツールで生成された異なる設定でのパフォーマンス見積もりを示します。

表 334: meanStdDev 関数のパフォーマンス見積もりのサマリ

動作モード	レイテンシ見積もり
	最大レイテンシ
1 ピクセル動作 (300 MHz)	6.9 ms
8 ピクセル動作 (150 MHz)	1.69 ms

## 最大 (Max)

Max 関数は、2 つの入力画像 src1 と src2 の各要素の最大値を計算し、結果を dst に保存します。

$$\text{dst}(x,y)=\max(\text{src1}(x,y),\text{src2}(x,y))$$

## API 構文

```
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)
```

## パラメーターの説明

次の表に、テンプレートと関数のパラメーターを説明します。

表 335: Max 関数のパラメーターの説明

パラメーター	説明
SRC_T	入力ピクセルのデータ型。8 ビット、符号なし、1 チャンネル (XF_8UC1) をサポート。
ROWS	入力および出力画像の最大高さ。
COLS	入力および出力画像の最大幅。N ピクセルの並列処理の場合、幅は N の倍数で指定。
NPC	1 サイクルごとに処理されるピクセル数。1 ピクセルの場合は XF_NPPC1、8 ピクセルの場合は XF_NPPC8。
_src1	1 つ目の入力画像
_src2	2 つ目の入力画像
_dst	出力画像

## リソース使用量

次の表に、Xczu9eg-ffvb1156-1-i-es1 FPGA で Vivado HLS 2019.1 バージョン ツールを使用して生成された、リソース最適化 (8 ピクセル) モードおよび通常モードにおける Max 関数のリソース使用量を示します。

表 336: Max 関数のリソース使用量のサマリ

名前	リソース使用量	
	1 クロックごとに 1 ピクセル動作	8 クロックごとに 1 ピクセル動作
	300 MHz	150 MHz
BRAM_18K	0	0
DSP48E	0	0
FF	103	153
LUT	44	102
CLB	21	38

## パフォーマンス見積もり

次の表に、Xczu9eg-ffvb1156-1-i-es1 FPGA でグレースケール HD (1080x1920) 画像を処理するために、Vivado HLS 2019.1 ツールを使用して生成された異なる設定でのカーネルのパフォーマンス見積もりを示します。

表 337: Max 関数のパフォーマンス見積もりのサマリ

動作モード	レイテンシ見積もり	
	動作周波数 (MHz)	レイテンシ (ms)
1 ピクセル	300	6.9
8 ピクセル	150	1.7

## MaxS

MaxS 関数は、src と指定のスカラー値を比較して、大きい方を dst に保存します。

$$\text{dst}(l) = \text{maxS}(\text{src}(l), \text{scl})$$

## API 構文

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

## パラメーターの説明

次の表に、テンプレートと関数のパラメーターを説明します。

表 338: MaxS 関数のパラメーターの説明

パラメーター	説明
SRC_T	入力ピクセルのデータ型。8 ビット、符号なし、1 チャンネル (XF_8UC1) をサポート。
ROWS	入力および出力画像の最大高さ。
COLS	入力および出力画像の最大幅。N ピクセルの並列処理の場合、幅は N の倍数で指定。
NPC	1 サイクルごとに処理されるピクセル数。1 ピクセルの場合は XF_NPPC1、8 ピクセルの場合は XF_NPPC8。
_src1	1 つ目の入力画像
_scl	入力スカラー値。サイズはチャンネル数にする必要があります。
_dst	出力画像

## リソース使用量

次の表に、Xczu9eg-ffvb1156-1-i-es1 FPGA で Vivado HLS 2019.1 バージョン ツールを使用して生成された、リソース最適化 (8 ピクセル) モードおよび通常モードにおける MaxS 関数のリソース使用量を示します。

表 339: MaxS 関数のリソース使用量のサマリ

名前	リソース使用量	
	1 クロックごとに 1 ピクセル動作	8 クロックごとに 1 ピクセル動作
	300 MHz	150 MHz
BRAM_18K	0	0
DSP48E	0	0
FF	162	43
LUT	103	104
CLB	32	20

## パフォーマンス見積もり

次の表に、Xczu9eg-ffvb1156-1-i-es1 FPGA でグレースケール HD (1080x1920) 画像を処理するために、Vivado HLS 2019.1 ツールを使用して生成された異なる設定でのカーネルのパフォーマンス見積もりを示します。

表 340: MaxS 関数のパフォーマンス見積もりのサマリ

動作モード	レイテンシ見積もり	
	動作周波数 (MHz)	レイテンシ (ms)
1 ピクセル	300	6.9

表 340: MaxS 関数のパフォーマンス見積もりのサマリ (続き)

動作モード	レイテンシ見積もり	
	動作周波数 (MHz)	レイテンシ (ms)
8 ピクセル	150	1.7

## メジアン フィルター (medianBlur)

medianBlur は、入力画像のメジアンフィルター処理を実行する関数です。メジアンフィルターは、ノイズ除去を改善する非線形のデジタル フィルターです。N サイズのフィルターの場合、各ピクセルに対して NxN 近隣ピクセル値の中央値が出力されます。

### API 構文

```
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)
```

### パラメーターの説明

次の表に、テンプレートと関数のパラメーターを説明します。

表 341: medianBlur 関数のパラメーターの説明

パラメーター	説明
FILTER_SIZE	ハードウェア カーネルを構築するハードウェア フィルターのウィンドウ サイズ。1 を超える奇数の正の整数を指定可能。
BORDER_TYPE	ハードウェア カーネルで処理される境界タイプ。現時点でサポートされているのは XF_BORDER_REPLICATE のみ。
TYPE	入力および出力のピクセルのデータ型。8 ビット、符号なし、1 および 3 チャンネル (XF_8UC1、XF_8UC3) のみサポート。
ROWS	処理される画像の行数。
COLS	処理される画像の列数。8 ピクセル動作の場合は 8 の倍数で指定。
NPC	並列で処理されるピクセル数。オプションは、XF_NPPC1 (1 クロックごとに 1 ピクセル処理の場合)、XF_NPPC8 (1 クロックごとに 8 ピクセル処理の場合)
_src	入力画像。
_dst	出力画像。

### リソース使用量

次の表に、Xczu9eg-ffvb1156-1-i-es1 FPGA 用に Vivado HLS 2019.1 バージョン ツールを使用して生成された、XF\_NPPC1 および XF\_NPPC8 設定の medianBlur 関数のリソース使用量を示します。

表 342: medianBlur 関数のリソース使用量のサマリ

動作モード	FILTER_SIZE	動作周波数 (MHz)	使用量の見積もり			
			LUT	FF	DSP	BRAM
1 ピクセル	3	300	1197	771	0	3
8 ピクセル	3	150	6559	1595	0	6
1 ピクセル	5	300	5860	1886	0	5

次の表に、Xczu9eg-ffvb1156-1-i-es1 FPGA 用に Vivado HLS 2019.1 バージョン ツールを使用して生成された、XF\_NPPC1 の 3 チャンネル画像入力の medianBlur 関数のリソース使用量を示します。

表 343: medianBlur 関数のリソース使用量のサマリ

動作モード	FILTER_SIZE	動作周波数 (MHz)	使用量の見積もり			
			LUT	FF	DSP	BRAM
1 ピクセル	3	300	2100	1971	0	9
1 ピクセル	5	300	13541	9720	0	15

### パフォーマンス見積もり

次の表に、サイリンクス Xczu9eg-ffvb1156-1-i-es1 FPGA 用に Vivado HLS 2019.1 バージョン ツールを使用して生成された medianBlur のパフォーマンス見積もりを示します。

表 344: medianBlur 関数のパフォーマンス見積もりのサマリ

動作モード	FILTER_SIZE	動作周波数 (MHz)	入力画像サイズ	レイテンシ見積もり
				最大 (ms)
1 ピクセル	3	300	1920x1080	6.99
8 ピクセル	3	150	1920x1080	1.75
1 ピクセル	5	300	1920x1080	7.00

## 最小 (Min)

Min 関数は、2 つの入力画像 src1 と src2 の各要素の最小値を計算し、結果を dst に保存します。

$$\text{dst}(l) = \min(\text{src1}(l), \text{src2}(l))$$

### API 構文

```
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)
```

## パラメーターの説明

次の表に、テンプレートと関数のパラメーターを説明します。

表 345: Min 関数のパラメーターの説明

パラメーター	説明
SRC_T	入力ピクセルのデータ型。8 ビット、符号なし、1 チャンネル (XF_8UC1) をサポート。
ROWS	入力および出力画像の最大高さ
COLS	入力および出力画像の最大幅。8 ピクセル動作の場合は 8 の倍数で指定。
NPC	1 サイクルごとに処理されるピクセル数。1 ピクセル動作の場合は XF_NPPC1、8 ピクセル動作の場合は XF_NPPC8。
_src1	1 つ目の入力画像
_src2	2 つ目の入力画像
_dst	出力画像

## リソース使用量

次の表に、Xczu9eg-ffvb1156-1-i-es1 FPGA で Vivado HLS 2019.1 バージョン ツールを使用して生成された、リソース最適化 (8 ピクセル) モードおよび通常モードにおける Min 関数のリソース使用量を示します。

表 346: Min 関数のリソース使用量のサマリ

名前	リソース使用量	
	1 クロックごとに 1 ピクセル動作	8 クロックごとに 1 ピクセル動作
	300 MHz	150 MHz
BRAM_18K	0	0
DSP48E	0	0
FF	103	153
LUT	44	102
CLB	23	34

## パフォーマンス見積もり

次の表に、Xczu9eg-ffvb1156-1-i-es1 FPGA でグレースケール HD (1080x1920) 画像を処理するために、Vivado HLS 2019.1 ツールを使用して生成された異なる設定でのカーネルのパフォーマンス見積もりを示します。

表 347: Min 関数のパフォーマンス見積もりのサマリ

動作モード	レイテンシ見積もり	
	動作周波数 (MHz)	レイテンシ (ms)
1 ピクセル	300	6.9
8 ピクセル	150	1.7

## MinS

MinS 関数は、src と指定のスカラー値を比較して、小さい方を dst に保存します。

$$\text{dst}(x,y)=\min S(\text{src}(x,y), \text{scl})$$

## API 構文

```
template< int SRC_T , int ROWS, int COLS, int NPC=1>
void MinS(xf::Mat<SRC_T, ROWS, COLS, NPC> & _src1, unsigned char
_dst1[XF_CHANNELS(SRC_T,NPC)], xf::Mat<SRC_T, ROWS, COLS, NPC> & _dst)
```

## パラメーターの説明

次の表に、テンプレートと関数のパラメーターを説明します。

表 348: MinS 関数のパラメーターの説明

パラメーター	説明
SRC_T	入力ピクセルのデータ型。8 ビット、符号なし、1 チャンネル (XF_8UC1) をサポート。
ROWS	入力および出力画像の最大高さ。
COLS	入力および出力画像の最大幅。N ピクセルの並列処理の場合、幅は N の倍数で指定。
NPC	1 サイクルごとに処理されるピクセル数。1 ピクセルの場合は XF_NPPC1、8 ピクセルの場合は XF_NPPC8。
_src1	1 つ目の入力画像
_scl	入力スカラー値。サイズはチャンネル数にする必要があります。
_dst	出力画像

## リソース使用量

次の表に、Xczu9eg-ffvb1156-1-i-es1 FPGA で Vivado HLS 2019.1 バージョン ツールを使用して生成された、リソース最適化 (8 ピクセル) モードおよび通常モードにおける MinS 関数のリソース使用量を示します。

表 349: MinS 関数のリソース使用量のサマリ

名前	リソース使用量	
	1 クロックごとに 1 ピクセル動作	8 クロックごとに 1 ピクセル動作
	300 MHz	150 MHz
BRAM_18K	0	0
DSP48E	0	0
FF	104	159
LUT	43	103
CLB	23	36

## パフォーマンス見積もり

次の表に、Xczu9eg-ffvb1156-1-i-es1 FPGA でグレースケール HD (1080x1920) 画像を処理するために、Vivado HLS 2019.1 ツールを使用して生成された異なる設定でのカーネルのパフォーマンス見積もりを示します。

表 350: MinS 関数のパフォーマンス見積もりのサマリ

動作モード	レイテンシ見積もり	
	動作周波数 (MHz)	レイテンシ (ms)
1 ピクセル	300	6.9
8 ピクセル	150	1.7

## MinMax ロケーション (minMaxLoc)

minMaxLoc 関数では、画像の最小値および最大値と、それらの値のロケーションが検出されます。

$$\begin{aligned}
 minVal &= \min_{\substack{0 \leq x' \leq width \\ 0 \leq y' \leq height}} src(x', y') \\
 maxVal &= \max_{\substack{0 \leq x' \leq width \\ 0 \leq y' \leq height}} src(x', y')
 \end{aligned}$$

### API 構文

```
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)
```

### パラメーターの説明

次の表に、テンプレートと関数のパラメーターを説明します。

表 351: minMaxLoc 関数のパラメーターの説明

パラメーター	説明
SRC_T	入力ピクセルのデータ型。8 ビット、符号なし、1 チャンネル (XF_8UC1)、16 ビット、符号なし、1 チャンネル (XF_16UC1)、16 ビット、符号付き、1 チャンネル (XF_16SC1)、32 ビット、符号付き、1 チャンネル (XF_32SC1) をサポート。
ROWS	処理される画像の行数。
COLS	処理される画像の列数。
NPC	1 サイクルごとに処理されるピクセル数。1 ピクセルの場合は XF_NPPC1、8 ピクセルの場合は XF_NPPC8。
_src	入力画像
max_val	int 型の画像の最大値。
min_val	int 型の画像の最小値。
_minlocx	最初の最小値の X 軸の位置。
_minlocy	最初の最小値の Y 軸の位置。
_maxlocx	最初の最大値の x 軸の位置。
_maxlocy	最初の最大値の Y 軸の位置。



## リソース使用量

次の表に、Xczu9eg-ffvb1156-1-i-es1 FPGA でグレースケール HD (1080x1920) 画像を処理するために、Vivado HLS 2019.1 ツールを使用して生成された minMaxLoc 関数のリソース使用量を示します。

表 352: minMaxLoc 関数のリソース使用量のサマリ

動作モード	動作周波数 (MHz)	使用量の見積もり				
		BRAM_18K	DSP_48E	FF	LUT	CLB
1 ピクセル	300	0	3	451	398	86
8 ピクセル	150	0	3	1049	1025	220

## パフォーマンス見積もり

次の表に、ザイリンクス Xczu9eg-ffvb1156-1-i-es1 でグレースケール HD (1080x1920) 画像を処理するために、Vivado HLS 2019.1 ツールで生成された異なる設定でのパフォーマンス見積もりを示します。

表 353: minMaxLoc 関数のパフォーマンス見積もりのサマリ

動作モード	レイテンシ見積もり
	最大レイテンシ
1 ピクセル動作 (300 MHz)	6.9 ms
8 ピクセル動作 (150 MHz)	1.69 ms

## 平均値シフト追跡 (MeanShift)

平均値シフト追跡は基本的な物体追跡アルゴリズムの 1 つで、前に初期化されたモデルと局所的に最も類似したビデオフレームのエリアを検索します。追跡する物体は、ヒストグラムで示されます。物体追跡アルゴリズムでは、ターゲットは主に矩形または楕円形の領域で示され、ターゲットモデルとターゲット候補が含まれます。物体の特性を示すため、色ヒストグラムが使用されます。ターゲットモデルは通常その確率密度関数 (PDF) で示されます。重み付き RGB ヒストグラムが使用され、物体のピクセル重要度が追加されます。

平均値シフト アルゴリズムは、密度関数の最大値を見つける反復手法です。物体追跡では、密度関数は追跡する物体とテストするフレームの色ヒストグラムを使用して形成される重み付き画像です。重み付きヒストグラムを使用すると、通常のヒストグラム計算とは異なり、空間的な位置も考慮されます。この関数は、入力画像ポインター、矩形物体の左上と右下の座標、フレーム数、および追跡ステータスを入力として取り込み、反復平均値シフト方法を使用した重心を返します。

## API 構文

```
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 );
```

## テンプレートのパラメーターの説明

次の表に、テンプレートのパラメーターを説明します。

表 354: MeanShift テンプレートのパラメーター

パラメーター	説明
MAXOBJ	追跡する物体の最大数
MAXITERS	最大反復数
OBJ_ROWS	追跡する物体の最大高さ
OBJ_COLS	追跡する物体の最大幅
SRC_T	入力 xf::Mat のデータ型。4 チャンネルの 8 ビット データ (XF_8UC4) のみサポート。
ROWS	画像の最大高さ
COLS	画像の最大幅
NPC	1 サイクルごとに処理されるピクセル数。XF_NPPC1 (1 サイクルごとに 1 ピクセル動作) のみサポート。

### 関数のパラメーターの説明

次の表に、関数のパラメーターを説明します。

表 355: MeanShift 関数のパラメーター

パラメーター	説明
_in_mat	入力 xF Mat
x1	すべての物体の左上角の X 座標
y1	すべての物体の左上角の Y 座標
obj_height	すべての物体の高さ
obj_width	すべての物体の幅
dx	カーネル関数で返されたすべての物体の X 軸の中心
dy	カーネル関数で返されたすべての物体の Y 軸の中心
status	物体のステータスが true の場合にのみ物体を追跡。物体がフレーム外の場合はステータスは 0。
frame_status	最初のフレームは 0、その他のフレームは 1 に設定。
no_objects	追跡する物体の数
no_iters	反復数。

### リソース使用量およびパフォーマンスの見積もり

次の表に、300 MHz の xczu9eg-ffvb1156-i-es1 で解像度 1920x1080、10 個の物体 (サイズ 250x250、4 反復) の RGB 画像を処理するために、Vivado HLS 2019.1 リリース ツールを使用して生成された標準 (1 ピクセル) 設定の MeanShift 関数のリソース使用量を示します。

表 356: MeanShift 関数のリソース使用量とパフォーマンス見積もりのサマリ

設定	最大レイテンシ (ms)	BRAM	DSP	FF	LUT
1 ピクセル	19.28	76	14	13198	10064

### 制限

追跡できる物体の最大数は 10 です。

## Otsu 法のしきい値処理 (OtsuThreshold)

Otsu 法のしきい値処理では、クラスタリング ベースの画像のしきい値処理またはグレースケール画像のバイナリ画像への縮小が自動的に実行されます。このアルゴリズムでは、2 クラスのピクセルの後に、バイモーダル ヒストグラム (前景ピクセルと背景ピクセル) が含まれ、その 2 つのクラスを分離する最適なしきい値が計算されます。

Otsu 法では、2 つのクラスを分離するクラス内分散 (2 つのクラスの分散の加重和で定義) を最小にできるしきい値を見つけます。

$$\sigma_w^2(t) = w_1 \sigma_1^2(t) + w_2 \sigma_2^2(t)$$

$w_1$  はヒストグラムから計算されたクラス確率です。

$$w_1 = \sum_{i=1}^t p(i) \quad w_2 = \sum_{i=t+1}^I p(i)$$

Otsu 法では、クラス内分散の最小化がクラス間分散の最大化と同じであることが示されます。

$$\sigma_b^2 = \sigma - \sigma_w^2$$

$$\sigma_b^2 = w_1 w_2 (\mu_b - \mu_f)^2$$

$$\mu_b = \left[ \sum_{i=1}^t p(i)x(i) \right] / w_1, \quad \mu_f = \left[ \sum_{i=t+1}^I p(i)x(i) \right] / w_2$$

はクラス平均です。

### API 構文

```
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)
```

### パラメーターの説明

次の表に、テンプレートと関数のパラメーターを説明します。

表 357: OtsuThreshold 関数のパラメーターの説明

パラメーター	説明
SRC_T	入力ピクセルのデータ型。8 ビット、符号なし、1 チャンネル (XF_8UC1) のみサポート。
ROWS	入力および出力画像の最大高さ。
COLS	入力および出力画像の最大幅 (8 ピクセル動作の場合は 8 の倍数で指定)
NPC	1 サイクルごとに処理されるピクセル数。1 ピクセルの場合は XF_NPPC1、8 ピクセルの場合は XF_NPPC8。
_src_mat	入力画像
_thresh	計算後の出力しきい値

## リソース使用量

次の表に、ザイリンクス Xczu9eg-ffvb1156-1-i-es1 FPGA でグレースケール HD (1080x1920) 画像を処理するために、Vivado HLS 2019.1 ツールを使用して生成された OtsuThreshold 関数のリソース使用量を示します。

表 358: OtsuThreshold 関数のリソース使用量のサマリ

動作モード	動作周波数 (MHz)	使用量の見積もり				
		BRAM_18K	DSP_48E	FF	LUT	CLB
1 ピクセル	300	8	49	2239	3353	653
8 ピクセル	150	22	49	1106	3615	704

## パフォーマンス見積もり

次の表に、ザイリンクス Xczu9eg-ffvb1156-1-i-es1 でグレースケール HD (1080x1920) 画像を処理するために、Vivado HLS 2019.1 ツールで生成された異なる設定でのパフォーマンス見積もりを示します。

表 359: OtsuThreshold 関数のパフォーマンス見積もりのサマリ

動作モード	レイテンシ見積もり
	最大レイテンシ (ms)
1 ピクセル動作 (300 MHz)	6.92 ms
8 ピクセル動作 (150 MHz)	1.76 ms

## paintmask

paintmask 関数は、入力画像のピクセルの強度値を、対応するマスクが 0 でない場合に指定の色に置き換えます。

### API 構文

```
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)])
```

### パラメーターの説明

次の表に、テンプレートと関数のパラメーターを説明します。

表 360: paintmask 関数のパラメーターの説明

パラメーター	説明
SRC_T	入力ピクセルのデータ型。8 ビット、符号なし、1 チャンネル (XF_8UC1) をサポート。
MASK_T	マスク値のデータ型。8 ビット、符号なし、1 チャンネル (XF_8UC1) をサポート。
ROWS	入力および出力画像の最大高さ
COLS	入力および出力画像の最大幅。N ピクセルの並列処理の場合、幅は N の倍数で指定。

表 360: paintmask 関数のパラメーターの説明 (続き)

パラメーター	説明
NPC	1 サイクルごとに処理されるピクセル数。1 ピクセルの場合は XF_NPPC1、8 ピクセルの場合は XF_NPPC8。
_src_mat	入力画像
_in_mask	入力マスク画像
_dst_mat	出力画像
_color	マスクが 0 でない場合に置き換える色の値

## リソース使用量

次の表に、Xczu9eg-ffvb1156-1-i-es1 FPGA で Vivado HLS 2019.1 バージョン ツールを使用して生成された、リソース最適化 (8 ピクセル) モードおよび通常モードにおける paintmask 関数のリソース使用量を示します。

表 361: paintmask 関数のリソース使用量のサマリ

名前	リソース使用量	
	1 クロックごとに 1 ピクセル動作	8 クロックごとに 1 ピクセル動作
	300 MHz	150 MHz
BRAM_18K	0	0
DSP48E	0	0
FF	95	163
LUT	57	121
CLB	14	33

## パフォーマンス見積もり

次の表に、Xczu9eg-ffvb1156-1-i-es1 FPGA でグレースケール HD (1080x1920) 画像を処理するために、Vivado HLS 2019.1 ツールを使用して生成された異なる設定でのカーネルのパフォーマンス見積もりを示します。

表 362: paintmask 関数のパフォーマンス見積もりのサマリ

動作モード	レイテンシ見積もり	
	動作周波数 (MHz)	レイテンシ (ms)
1 ピクセル	300	6.9
8 ピクセル	150	1.7

## ピクセル加算 (add)

add 関数は、2 つの入力画像間でピクセル単位の加算を実行して、出力画像を返します。

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

説明:

- $I_{out}(x, y)$ : 出力画像の (x,y) 位置での強度。
- $I_{in1}(x, y)$ : 最初の入力画像の (x,y) 位置での強度。

- $I_{in2}(x, y)$ : 2 つ目の入力画像の  $(x, y)$  位置での強度。

XF\_CONVERT\_POLICY\_TRUNCATE: 結果は出力オペランドの LSB で、そのビット深さのサイズの 2 の補数バイナリフォーマットで保存されます。

XF\_CONVERT\_POLICY\_SATURATE: 結果は出力オペランドのビットの深さに飽和されます。

## API 構文

```
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 )
```

## パラメーターの説明

次の表に、テンプレートと関数のパラメーターを説明します。

表 363: add 関数のパラメーターの説明

パラメーター	説明
POLICY_TYPE	オーバーフロー処理のタイプ。XF_CONVERT_POLICY_SATURATE または XF_CONVERT_POLICY_TRUNCATE のいずれかに設定可能。
SRC_T	ピクセルのデータ型。オプションは XF_8UC1、XF_8UC3、XF_16SC3、および XF_16SC1。
ROWS	入力および出力画像の最大高さ。
COLS	入力および出力画像の最大幅 (8 ピクセル動作の場合は 8 の倍数で指定)
NPC	1 サイクルごとに処理されるピクセル数。1 ピクセルの場合は XF_NPPC1、8 ピクセルの場合は XF_NPPC8。
src1	入力画像
src2	入力画像
dst	出力画像

## リソース使用量

次の表に、サイリンクス Xczu9eg-ffvb1156-1-i-es1 でグレースケール HD (1080x1920) 画像を処理するために、Vivado HLS 2019.1 ツールで生成された異なる設定でのリソース使用量を示します。

表 364: add 関数のリソース使用量のサマリ

動作モード	動作周波数 (MHz)	使用量の見積もり				
		BRAM_18K	DSP_48E	FF	LUT	CLB
1 ピクセル	300	0	0	62	55	11
8 ピクセル	150	0	0	65	138	24

次の表に、サイリンクス Xczu9eg-ffvb1156-1-i-es1 FPGA で 3 チャンネルの 4K 画像を処理するために、Vivado HLS 2019.1 ツールで生成された異なる設定でのリソース使用量を示します。

表 365: add 関数のリソース使用量のサマリ

動作モード	動作周波数 (MHz)	使用量の見積もり				
		BRAM_18K	DSP_48E	FF	LUT	CLB
1 ピクセル	300	0	0	113	77	24

### パフォーマンス見積もり

次の表に、ザイリンクス Xczu9eg-ffvb1156-1-i-es1 でグレースケール HD (1080x1920) 画像を処理するために、Vivado HLS 2019.1 ツールで生成された異なる設定でのパフォーマンス見積もりを示します。

表 366: add 関数のパフォーマンス見積もりのサマリ

動作モード	レイテンシ見積もり
	最大レイテンシ (ms)
1 ピクセル動作 (300 MHz)	6.9
8 ピクセル動作 (150 MHz)	1.7

## ピクセル乗算 (multiply)

multiply 関数は、2 つの入力画像間でピクセル単位の乗算を実行して、出力画像を返します。

$$I_{out}(x, y) = I_{in1}(x, y) * I_{in2}(x, y) * scale\_val$$

説明:

- $I_{out}(x, y)$ : 出力画像の (x,y) 位置での強度。
- $I_{in1}(x, y)$ : 最初の入力画像の (x,y) 位置での強度。
- $I_{in2}(x, y)$ : 2 つ目の入力画像の (x,y) 位置での強度
- $scale\_val$ : スケール値。

XF\_CONVERT\_POLICY\_TRUNCATE: 結果は出力オペランドの LSB で、そのビット深さのサイズの 2 の補数バイナリフォーマットで保存されます。

XF\_CONVERT\_POLICY\_SATURATE: 結果は出力オペランドのビットの深さに飽和されます。

### API 構文

```
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)
```

### パラメーターの説明

次の表に、テンプレートと関数のパラメーターを説明します。

表 367: multiply 関数のパラメーターの説明

パラメーター	説明
POLICY_TYPE	オーバーフロー処理のタイプ。XF_CONVERT_POLICY_SATURATE または XF_CONVERT_POLICY_TRUNCATE のいずれかに設定可能。
SRC_T	ピクセルのデータ型。オプションは XF_8UC1、XF_8UC3、XF_16SC1、および XF_16SC3。
ROWS	入力および出力画像の最大高さ。
COLS	入力および出力画像の最大幅 (8 ピクセル動作の場合は 8 の倍数で指定)
NPC	1 サイクルごとに処理されるピクセル数。1 ピクセルの場合は XF_NPPC1、8 ピクセルの場合は XF_NPPC8。
src1	入力画像
src2	入力画像
dst	出力画像
scale_val	0 ～ 1 の重み係数

### リソース使用量

次の表に、ザイリンクス Xczu9eg-ffvb1156-1-i-es1 でグレースケール HD (1080x1920) 画像を処理するために、Vivado HLS 2019.1 ツールで生成された異なる設定でのリソース使用量を示します。

表 368: multiply 関数のリソース使用量のサマリ

動作モード	動作周波数 (MHz)	使用量の見積もり				
		BRAM_18K	DSP_48E	FF	LUT	CLB
1 ピクセル	300	0	2	124	59	18
8 ピクセル	150	0	16	285	108	43

次の表に、ザイリンクス Xczu9eg-ffvb1156-1-i-es1 FPGA で 3 チャンネルの 4K 画像を処理するために、Vivado HLS 2019.1 ツールで生成された異なる設定でのリソース使用量を示します。

表 369: multiply 関数のリソース使用量のサマリ

動作モード	動作周波数 (MHz)	使用量の見積もり				
		BRAM_18K	DSP_48E	FF	LUT	CLB
1 ピクセル	300	0	9	312	211	62

### パフォーマンス見積もり

次の表に、ザイリンクス Xczu9eg-ffvb1156-1-i-es1 でグレースケール HD (1080x1920) 画像を処理するために、Vivado HLS 2019.1 ツールで生成された異なる設定でのパフォーマンス見積もりを示します。



表 370: multiply 関数のパフォーマンス見積りサマリ

動作モード	レイテンシ見積り
	最大レイテンシ (ms)
1 ピクセル動作 (300 MHz)	6.9
8 ピクセル動作 (150 MHz)	1.6

## ピクセル減算 (subtract)

subtract 関数は、2 つの入力画像間でピクセル単位の減算を実行して、出力画像を返します。

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

説明:

- $I_{out}(x, y)$ : 出力画像の (x,y) 位置での強度。
- $I_{in1}(x, y)$ : 最初の入力画像の (x,y) 位置での強度。
- $I_{in2}(x, y)$ : 2 つ目の入力画像の (x,y) 位置での強度。

XF\_CONVERT\_POLICY\_TRUNCATE: 結果は出力オペランドの LSB で、そのビット深さのサイズの 2 の補数バイナリフォーマットで保存されます。

XF\_CONVERT\_POLICY\_SATURATE: 結果は出力オペランドのビットの深さに飽和されます。

### API 構文

```
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 )
```

### パラメーターの説明

次の表に、テンプレートと関数のパラメーターを説明します。

表 371: subtract 関数のパラメーターの説明

パラメーター	説明
POLICY_TYPE	オーバーフロー処理のタイプ。XF_CONVERT_POLICY_SATURATE または XF_CONVERT_POLICY_TRUNCATE のいずれかに設定可能。
SRC_T	ピクセルのデータ型。オプションは XF_8UC1、XF_8UC3、XF_16SC3、および XF_16SC1。
ROWS	入力および出力画像の最大高さ。
COLS	入力および出力画像の最大幅 (8 ピクセル動作の場合は 8 の倍数で指定)
NPC	1 サイクルごとに処理されるピクセル数。1 ピクセルの場合は XF_NPPC1、8 ピクセルの場合は XF_NPPC8。
src1	入力画像
src2	入力画像
dst	出力画像

## リソース使用量

次の表に、ザイリンクス Xczu9eg-ffvb1156-1-i-es1 でグレースケール HD (1080x1920) 画像を処理するために、Vivado HLS 2019.1 ツールで生成された異なる設定でのリソース使用量を示します。

表 372: **subtract** 関数のリソース使用量のサマリ

動作モード	動作周波数 (MHz)	使用量の見積もり				
		BRAM_18K	DSP_48E	FF	LUT	CLB
1 ピクセル	300	0	0	62	53	11
8 ピクセル	150	0	0	59	13	21

次の表に、ザイリンクス Xczu9eg-ffvb1156-1-i-es1 FPGA で 3 チャンネルの 4K 画像を処理するために、Vivado HLS 2019.1 ツールで生成された異なる設定でのリソース使用量を示します。

表 373: **subtract** 関数のリソース使用量のサマリ

動作モード	動作周波数 (MHz)	使用量の見積もり				
		BRAM_18K	DSP_48E	FF	LUT	CLB
1 ピクセル	300	0	0	110	64	28

## パフォーマンス見積もり

次の表に、ザイリンクス Xczu9eg-ffvb1156-1-i-es1 でグレースケール HD (1080x1920) 画像を処理するために、Vivado HLS 2019.1 ツールで生成された異なる設定でのパフォーマンス見積もりを示します。

表 374: **subtract** 関数のパフォーマンス見積もりのサマリ

動作モード	レイテンシ見積もり
	最大レイテンシ (ms)
1 ピクセル動作 (300 MHz)	6.9
8 ピクセル動作 (150 MHz)	1.7

## Reduce

reduce 関数は、行/列を 1-D ベクターのセットとして扱い、1 つの行/列が得られるまで指定の演算を実行することにより、行列をベクターに簡約します。

簡約演算は、次のいずれかになります。

- REDUCE\_SUM: 行列の行と列すべての合計を出力します。
- REDUCE\_AVG: 行列の行と列すべての平均ベクターを出力します。
- REDUCE\_MAX: 行列の行と列すべてで最大のもの (列方向/行方向) を出力します。
- REDUCE\_MIN: 行列の行と列すべてで最小のもの (列方向/行方向) を出力します。

## API 構文

```
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)
```

## パラメーターの説明

次の表に、テンプレートと関数のパラメーターを説明します。

表 375: reduce 関数のパラメーターの説明

パラメーター	説明
REDUCE_OP	適用する簡約演算のタイプを指定するフラグ。
SRC_T	入力ピクセルのデータ型。8 ビット、符号なし、1 チャンネル (XF_8UC1) をサポート。
DST_T	出力ピクセルのデータ型。8 ビット、符号なし、1 チャンネル (XF_8UC1) をサポート。
ROWS	入力および出力画像の最大高さ
COLS	入力および出力画像の最大幅。N ピクセルの並列処理の場合、幅は N の倍数で指定。
ONE_D_HEIGHT	出力 1-D ベクターまたは簡約された行列の高さ
ONE_D_WIDTH	出力 1-D ベクターまたは簡約された行列の幅
NPC	1 サイクルごとに処理されるピクセル数。可能なオプションは XF_NPPC1 (1 サイクルごとに 1 ピクセル)。
_src_mat	入力画像
_dst_mat	1-D ベクター
dim	行列を簡約する次元指数。0 は行列を 1 行に簡約、1 は行列を 1 列に簡約することを指定します。

## リソース使用量

次の表に、Xczu9eg-ffvb1156-1-i-es1 FPGA で Vivado HLS 2019.1 バージョン ツールを使用して生成された通常モード (1 ピクセル) での reduce 関数のリソース使用量を示します。

表 376: reduce 関数のリソース使用量のサマリ

名前	リソース使用量
	1 クロックごとに 1 ピクセル動作
	300 MHz
BRAM_18K	2
DSP48E	0
FF	288
LUT	172
CLB	54

## パフォーマンス見積もり

次の表に、Xczu9eg-ffvb1156-1-i-es1 FPGA でグレースケール HD (1080x1920) 画像を処理するために、Vivado HLS 2019.1 ツールを使用して生成された異なる設定でのカーネルのパフォーマンス見積もりを示します。

表 377: reduce 関数のパフォーマンス見積もりのサマリ

動作モード	レイテンシ見積もり	
	動作周波数 (MHz)	レイテンシ (ms)
1 ピクセル	300	6.9

## リマップ (remap)

remap 関数は、画像内の 1 つの位置からピクセルを取り出して、別の画像の別の位置に移動します。移動前の画像から移動後の画像へのマップには、2 種類の補間方法が使用されます。

$$dst = src(map_x(x, y), map_y(x, y))$$

### API 構文

```
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);
```

### パラメーターの説明

次の表に、テンプレートのパラメーターを説明します。

表 378: remap テンプレートのパラメーターの説明

パラメーター	説明
WIN_ROWS	内部でバッファリングされる入力画像の行数。マップデータに基づいて設定する必要あり。たとえば、左右反転には 2 行で十分。
INTERPOLATION_TYPE	補間タイプは XF_INTERPOLATION_NN (最近傍補間) または XF_INTERPOLATION_BILINEAR (バイリニア補間) のいずれか。
SRC_T	入力および出力のピクセルのデータ型。8 ビット、符号なし、1 および 3 チャンネル (XF_8UC1、XF_8UC3) のみサポート。
MAP_T	マップタイプ。1 チャンネルの浮動小数点型。XF_32FC1。
DST_T	出力画像タイプ。8 ビットで 1 チャンネルのグレースケール画像。XF_8UC1。
ROWS	入力および出力画像の高さ。
COLS	入力および出力画像の幅。
NPC	1 サイクルごとに処理されるピクセル数。XF_NPPC1 (1 サイクルごとに 1 ピクセル動作) のみサポート。
USE_URAM	一部の構造をブロック RAM ではなく UltraRAM にマップ。

次の表に、関数のパラメーターを説明します。

表 379: remap 関数のパラメーターの説明

PARAMETERS	説明
_src_mat	入力 xF Mat
_remapped_mat	出力 xF Mat
_mapx_mat	浮動小数点型の mapX Mat
_mapy_mat	浮動小数点型の mapY Mat

## リソース使用量

次の表に、xczu9eg-ffvb1156-i-es1 FPGA 用に 300 MHz、XF\_INTERPOLATION\_BILINEAR モードの WIN\_ROWS を 64 で Vivado HLS 2019.1 ツールを使用して生成された HD (1080x1920) 画像の remap のリソース使用量を示します。

表 380: remap 関数のリソース使用量のサマリ

名前	リソース使用量
BRAM_18K	64
DSP48E	17
FF	1738
LUT	1593
CLB	360

次の表に、ザイリンクス xczu7ev-ffvc1156 FPGA 用に SDx 2019.1 ツールを使用して 300 MHz で XF\_INTERPOLATION\_BILINEAR モードの WIN\_ROWS を 100 に設定して生成された 4K (3840x2160) 画像の remap のリソース使用量を示します。

表 381: UltraRAM をイネーブルにした場合の remap 関数のリソース使用量のサマリ

名前	リソース使用量
BRAM_18K	3
DSP48E	10
URAM	24
FF	3196
LUT	3705

## パフォーマンス見積もり

次の表に、xczu9eg-ffvb1156-i-es1 FPGA 用に 300 MHz、XF\_INTERPOLATION\_BILINEAR モードの WIN\_ROWS 64 で Vivado HLS 2019.1 ツールを使用して生成された HD (1080x1920) 画像の remap() のパフォーマンス見積もりを示します。

表 382: remap 関数のパフォーマンス見積もりのサマリ

動作モード	動作周波数 (MHz)	レイテンシ見積もり 最大レイテンシ (ms)
1 ピクセル モード	300	7.2

## 解像度変換/リサイズ (resize)

解像度変換は、元の画像のサイズをターゲットの画像サイズに変換するのに使用される方法です。リサイズ関数には、最近傍補間、バイリニア補間、エリア補間など、さまざまな補間方法を使用できます。補間タイプは、API にテンプレートパラメーターとして渡すことができます。指定可能な補間タイプは次のとおりです。

- XF\_INTERPOLATION\_NN: 最近傍補間
- XF\_INTERPOLATION\_BILINEAR: バイリニア補間
- XF\_INTERPOLATION\_AREA: エリア補間

**注記:** ダウンスケールには 0.25 以上、アップスケールには 8 以下のスケール係数がサポートされます。

### API 構文

```
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)
```

### パラメーターの説明

次の表に、テンプレートと関数のパラメーターを説明します。

表 383: **resize** 関数のパラメーターの説明

パラメーター	説明
INTERPOLATION_TYPE	補間タイプ。使用可能なオプションは次のとおりです。 <ul style="list-style-type: none"> <li>• XF_INTERPOLATION_NN: 最近傍補間</li> <li>• XF_INTERPOLATION_BILINEAR: バイリニア補間</li> <li>• XF_INTERPOLATION_AREA: エリア補間</li> </ul>
TYPE	入力および出力のピクセルのデータ型。8 ビット、符号なし、1 および 3 チャンネル (XF_8UC1、XF_8UC3) のみサポート。
SRC_ROWS	ハードウェア カーネルを構築する入力画像の最大高さ。
SRC_COLS	ハードウェア カーネルを構築する入力画像の最大幅 (8 の倍数を指定)。
DST_ROWS	ハードウェア カーネルを構築する出力画像の最大高さ。
DST_COLS	ハードウェア カーネルを構築する出力画像の最大幅 (8 の倍数を指定)。
NPC	1 サイクルごとに処理されるピクセル数。可能なオプションは XF_NPPC1 (1 サイクルごとに 1 ピクセル) および XF_NPPC8 (1 サイクルごとに 8 ピクセル)。
MAX_DOWN_SCALE	1 ピクセル モードでの x 方向へのアップスケールの場合は 2 に設定します。8 ピクセル モードでの x 方向へのダウンスケールでは、ダウンスケール係数の次に大きい整数値に指定します。たとえば、1920 列を 1280 列にダウンスケールする場合は 2 に設定し、1920 列を 640 列にダウンスケールする場合は 3 に設定します。
_src	入力画像
_dst	出力画像

### リソース使用量

次の表に、xczu9eg-ffvb1156-2-i-es2 FPGA 用に Vivado HLS 2019.1 ツールを使用して生成された Resource Optimized (8 ピクセル) モードと Normal モードの **resize** 関数のリソース使用量を示します。

表 384: **resize** 関数のリソース使用量のサマリ

動作モード	使用量の見積もり									
	1 ピクセル (300 MHz)					8 ピクセル (150 MHz)				
	IMAGESIZE	LUT	FF	DSP	BRAM	IMAGESIZE	LUT	FF	DSP	BRAM
ダウンスケール最近傍	1920X1080 → 960X1620	1089	1593	4	2	3840X2160 → 1920X1080	2545	2250	4	12
ダウンスケール バイリニア	1920X1080 → 960X1080	1340	1846	8	2	3840X2160 → 1920X1080	5159	3092	36	12
ダウンスケール エリア	3840X2160 → 1920X1080	2341	3550	44	24	サポートなし				
アップスケール最近傍	1920X1080 → 3840X540	1089	1593	4	2	1920X1080 → 3840X2160	1818	1686	4	6
アップスケール バイリニア	1920X1080 → 3840X540	1340	1846	8	2	1920X1080 → 3840X2160	3697	2739	36	6
アップスケール エリア	1920X1080 → 3840X2160	1312	2220	16	12	サポートなし				

次の表に、3 チャンネル画像入力を処理するために xczu9eg-ffvb1156-2-i-es2 FPGA 用に Vivado HLS 2019.1 ツールを使用して生成された通常モードの **resize** 関数のリソース使用量を示します。

表 385: **resize** 関数のリソース使用量のサマリ

動作モード	使用量の見積もり				
	1 ピクセル (300 MHz)				
	IMAGESIZE	LUT	FF	DSP	BRAM
ダウンスケール最近傍	3840X2160 → 1920X108	1184	168	4	18
ダウンスケール バイリニア	3840X2160 → 1920X1080	1592	2058	14	18
ダウンスケール エリア	3840X2160 → 1920X1080	3212	4777	104	72
アップスケール最近傍	1920X1080 → 3840X2160	1166	1697	4	9
アップスケール バイリニア	1920X1080 → 3840X2160	1574	2053	14	9
アップスケール エリア	1920X1080 → 3840X2160	1731	2733	36	31

### パフォーマンス見積もり

次の表に、300 MHz の xczu9eg-ffvb1156-2-i-es2 FPGA でグレースケール画像を 1080x1920 から 480x640 にリサイズ (ダウンスケール) し、1080x1920 から 2160x3840 にリサイズ (アップスケール) するために Vivado HLS 2019.1 ツールを使用して生成されたさまざまな設定のリサイズ関数のパフォーマンス見積もりを示します。補間タイプ別のレイテンシも示します。

表 386: **resize** 関数のパフォーマンス見積もりのサマリ

動作モード	動作周波数 (MHz)	レイテンシ見積もり (ms)					
		ダウン スケール 最近傍 (NN)	ダウン スケール バイリニア	ダウン スケール エリア	アップ スケール 最近傍 (NN)	アップ スケール バイリニア	アップ スケール エリア
1 ピクセル	300	6.94	6.97	7.09	27.71	27.75	27.74

## BGR2HSV

BGR2HSV 関数は、入力画像の色空間を HSV 色空間に変換し、出力として HSV 画像を返します。

### API 構文

```
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)
```

### パラメーターの説明

次の表に、テンプレートと関数のパラメーターを説明します。

パラメーター	説明
SRC_T	入力ピクセルのデータ型。XF_8UC3 のみサポート。
DST_T	出力ピクセルのデータ型は XF_8UC3 である必要あり
ROWS	入力および出力画像の最大高さ
COLS	入力および出力画像の最大幅。8 ピクセル動作の場合は 8 の倍数で指定。
NPC	1 サイクルごとに処理されるピクセル数。XF_NPPC1 のみサポート。
_src_mat	入力画像
_dst_mat	出力画像

## convertScaleAbs

convertScaleAbs 関数は、入力画像 src をオプションの線形変換を使用して変換し、結果を画像 dst に保存します。

$$\text{dst}(x,y) = \text{src1}(x,y) * \text{scale} + \text{shift}$$

### API 構文

```
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)
```

### パラメーターの説明

次の表に、テンプレートと関数のパラメーターを説明します。

表 387: convertScaleAbs 関数のパラメーターの説明

パラメーター	説明
SRC_T	入力ピクセルのデータ型。8 ビット、符号なし、1 チャンネル (XF_8UC1) のみサポート。
DST_T	出力ピクセルのデータ型。8 ビット、符号なし、1 チャンネル (XF_8UC1) のみサポート。
ROWS	入力および出力画像の最大高さ。
COLS	入力および出力画像の最大幅。N ピクセルの並列処理の場合、幅は N の倍数で指定。
NPC	1 サイクルごとに処理されるピクセル数。1 ピクセルの場合は XF_NPPC1、8 ピクセルの場合は XF_NPPC8。



表 387: convertScaleAbs 関数のパラメーターの説明 (続き)

パラメーター	説明
src1	入力画像
scale	スケール係数
shift	スケール値に追加するデルタ/シフト。
dst	出力画像

## リソース使用量

次の表に、Xczu9eg-ffvb1156-1-i-es1 FPGA で Vivado HLS 2019.1 バージョン ツールを使用して生成された、リソース最適化 (8 ピクセル) モードおよび通常モードにおける convertScaleAbs 関数のリソース使用量を示します。

表 388: convertScaleAbs 関数のリソース使用量のサマリ

名前	リソース使用量	
	1 クロックごとに 1 ピクセル動作	8 クロックごとに 1 ピクセル動作
	300 MHz	150 MHz
BRAM_18K	0	0
DSP48E	10	38
FF	949	1971
LUT	1052	1522
CLB	218	382

## パフォーマンス見積もり

次の表に、Xczu9eg-ffvb1156-1-i-es1 FPGA でグレースケール HD (1080x1920) 画像を処理するために、Vivado HLS 2019.1 ツールを使用して生成された異なる設定でのカーネルのパフォーマンス見積もりを示します。

表 389: convertScaleAbs 関数のパフォーマンス見積もりのサマリ

動作モード	レイテンシ見積もり	
	動作周波数 (MHz)	レイテンシ (ms)
1 ピクセル	300	6.9
8 ピクセル	150	1.7

## Scharr フィルター (Scharr)

Scharr 関数では、処理される入力画像でカーネルをたたみ込むことで、x と y の両方の方向で入力画像の勾配が計算されます。

カーネル サイズ 3x3 の場合:

- 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 構文

```
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)
```

## パラメーターの説明

次の表に、テンプレートと関数のパラメーターを説明します。

表 390: Scharr 関数のパラメーターの説明

パラメーター	説明
BORDER_TYPE	サポートされる境界タイプは XF_BORDER_CONSTANT。
SRC_T	入力ピクセルのデータ型。8 ビット、符号なし、1 および 3 チャンネル (XF_8UC1、XF_8UC3) のみサポート。
DST_T	出力ピクセルのデータ型。8 ビット符号なし、16 ビット符号付き、1 および 3 チャンネル (XF_8UC1、XF_16SC1、XF_8UC3) のみサポート。
ROWS	入力および出力画像の最大高さ
COLS	入力および出力画像の最大幅。8 ピクセル動作の場合は 8 の倍数で指定。
NPC	1 サイクルごとに処理されるピクセル数。1 ピクセルの場合は XF_NPPC1、8 ピクセルの場合は XF_NPPC8。
_src_mat	入力画像
_dst_matx	X 勾配の出力画像。
_dst_maty	Y 勾配の出力画像。

## リソース使用量

次の表に、ザイリンクス Xczu9eg-ffvb1156-1-i-es1 FPGA でグレースケール HD (1080x1920) 画像を処理するために、Vivado HLS 2019.1 ツールを使用して生成された異なる設定でのカーネルのリソース使用量を示します。

表 391: Scharr 関数のリソース使用量のサマリ

名前	リソース使用量	
	1 ピクセル	8 ピクセル
	300 MHz	150 MHz
BRAM_18K	3	6
DSP48E	0	0
FF	728	1434
LUT	812	2481
CLB	171	461

次の表に、ザイリンクス Xczu9eg-ffvb1156-1-i-es1 FPGA で 3 チャンネルの 4K 画像を処理するために、Vivado HLS 2019.1 ツールを使用して生成された異なる設定でのカーネルのリソース使用量を示します。

表 392: Scharr 関数のリソース使用量のサマリ

名前	リソース使用量
	1 ピクセル
	300 MHz
BRAM_18K	18
DSP48E	0
FF	1911
LUT	1392

### パフォーマンス見積もり

次の表に、ザイリンクス Xczu9eg-ffvb1156-1-i-es1 でグレースケール HD (1080x1920) 画像を処理するために、Vivado HLS 2019.1 ツールを使用して生成された異なる設定でのカーネルのパフォーマンス見積もりを示します。

表 393: Scharr 関数のパフォーマンス見積もりのサマリ

動作モード	動作周波数 (MHz)	レイテンシ (ms)
1 ピクセル	300	7.2
8 ピクセル	150	1.7

## 設定 (set)

set 関数は、入力画像の各ピクセルを指定のスカラー値に設定し、結果を dst に保存します。

### API 構文

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

### パラメーターの説明

次の表に、テンプレートと関数のパラメーターを説明します。

表 394: set 関数のパラメーターの説明

パラメーター	説明
SRC_T	入力ピクセルのデータ型。8 ビット、符号なし、1 チャンネル (XF_8UC1) をサポート。
ROWS	入力および出力画像の最大高さ
COLS	入力および出力画像の最大幅。8 ピクセル動作の場合は 8 の倍数で指定。
NPC	1 サイクルごとに処理されるピクセル数。

表 394: set 関数のパラメーターの説明 (続き)

パラメーター	説明
_src1	1 つ目の入力画像
_scl	スカラー値
_dst	出力画像

## リソース使用量

次の表に、Xczu9eg-ffvb1156-1-i-es1 FPGA で Vivado HLS 2019.1 バージョン ツールを使用して生成された、リソース最適化 (8 ピクセル) モードおよび通常モードにおける set 関数のリソース使用量を示します。

表 395: set 関数のリソース使用量のサマリ

名前	リソース使用量	
	1 クロックごとに 1 ピクセル動作	8 クロックごとに 1 ピクセル動作
	300 MHz	150 MHz
BRAM_18K	0	0
DSP48E	0	0
FF	87	87
LUT	43	42
CLB	17	18

## パフォーマンス見積もり

次の表に、Xczu9eg-ffvb1156-1-i-es1 FPGA でグレースケール HD (1080x1920) 画像を処理するために、Vivado HLS 2019.1 ツールを使用して生成された異なる設定でのカーネルのパフォーマンス見積もりを示します。

表 396: set 関数のパフォーマンス見積もりのサマリ

動作モード	レイテンシ見積もり	
	動作周波数 (MHz)	レイテンシ (ms)
1 ピクセル	300	6.9
8 ピクセル	150	1.7

## Sobel フィルター (Sobel)

Sobel 関数では、処理される入力画像でカーネルをたたみ込むことで、x と y の両方の方向で入力画像の勾配が計算されます。

- カーネル サイズ 3x3 の場合:

- GradientX:

$$G_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} * I$$

- GradientY:

$$G_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix} * I$$

- カーネルサイズ 5x5 の場合:

- 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:

$$G_y = \begin{bmatrix} -1 & -4 & -6 & -4 & -1 \\ -2 & -8 & -12 & -8 & -2 \\ 0 & 0 & 0 & 0 & 0 \\ 2 & 8 & 12 & 8 & 2 \\ 1 & 4 & 6 & 4 & 1 \end{bmatrix} * I$$

- カーネルサイズ 7x7 の場合:

- 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 構文

```
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)
```

## パラメーターの説明

次の表に、テンプレートと関数のパラメーターを説明します。

表 397: Sobel 関数のパラメーターの説明

パラメーター	説明
FILTER_TYPE	フィルター サイズ。サポートされるフィルター サイズは 3 (XF_FILTER_3X3)、5 (XF_FILTER_5X5)、および 7 (XF_FILTER_7X7)。
BORDER_TYPE	サポートされる境界タイプは XF_BORDER_CONSTANT。
SRC_T	入力ピクセルのデータ型。8 ビット、符号なし、1 および 3 チャンネル (XF_8UC1、XF_8UC3) のみサポート。
DST_T	出力ピクセルのデータ型。8 ビット符号なし、16 ビット符号付き、1 および 3 チャンネル (XF_8UC1、XF_16SC1、XF_8UC3) のみサポート。
ROWS	入力および出力画像の最大高さ。
COLS	入力および出力画像の最大幅。8 ピクセル動作の場合は 8 の倍数で指定。
NPC	1 サイクルごとに処理されるピクセル数。1 ピクセルの場合は XF_NPPC1、8 ピクセルの場合は XF_NPPC8。
USE_URAM	ストレージ構造を UltraRAM にマップ
_src_mat	入力画像
_dst_matx	X 勾配の出力画像。
_dst_maty	Y 勾配の出力画像。
1. Sobel 7x7 8 ピクセルはサポートされません。	

## リソース使用量

次の表に、ザイリンクス Xczu9eg-ffvb1156-1-i-es1 FPGA でグレースケール HD (1080x1920) 画像を処理するために、Vivado HLS 2019.1 ツールを使用して生成された異なる設定でのカーネルのリソース使用量を示します。

表 398: Sobel 関数のリソース使用量のサマリ

動作モード	フィルター サイズ	動作周波数 (MHz)	使用量の見積もり				
			BRAM_18K	DSP_48E	FF	LUT	CLB
1 ピクセル	3x3	300	3	0	609	616	135
	5x5	300	5	0	1133	1499	308
	7x7	300	7	0	2658	3334	632
8 ピクセル	3x3	150	6	0	1159	1892	341
	5x5	150	10	0	3024	5801	999

次の表に、ザイリンクス Xczu9eg-ffvb1156-1-i-es1 FPGA で 3 チャンネルの 4K 画像を処理するために、Vivado HLS 2019.1 ツールを使用して生成された異なる設定でのカーネルのリソース使用量を示します。

表 399: Sobel 関数のリソース使用量のサマリ

動作モード	フィルター サイズ	動作周波数 (MHz)	使用量の見積もり			
			BRAM_18K	DSP_48E	FF	LUT
1 ピクセル	3x3	300	18	0	1047	1107
	5x5	300	30	0	5370	3312
	7x7	300	42	0	6100	5496

次の表に、ザイリンクス xczu7ev-ffvc1156-2-e FPGA で UltraRAM をイネーブルにしてグレースケール 4K (3840x2160) 画像を処理するために、SDx 2019.1 ツールを使用して生成された異なる設定でのカーネルのリソース使用量を示します。

表 400: UltraRAM をイネーブルにした場合の Sobel 関数のリソース使用量のサマリ

動作モード	フィルター サイズ	動作周波数 (MHz)	使用量の見積もり				
			BRAM_18K	URAM	DSP_48E	FF	LUT
1 ピクセル	3x3	300	0	1	0	919	707
	5x5	300	0	1	0	2440	1557
	7x7	300	0	1	0	4066	3495
8 ピクセル	3x3	150	0	3	0	1803	2050
	5x5	150	0	5	0	4159	6817

### パフォーマンス見積もり

次の表に、ザイリンクス Xczu9eg-ffvb1156-1-i-es1 でグレースケール HD (1080x1920) 画像を処理するために、Vivado HLS 2019.1 ツールを使用して生成された異なる設定でのカーネルのパフォーマンス見積もりを示します。

表 401: Sobel 関数のパフォーマンス見積もりのサマリ

動作モード	動作周波数 (MHz)	フィルター サイズ	レイテンシ見積もり (ms)
1 ピクセル	300	3x3	7.5
	300	5x5	7.5
	300	7x7	7.5
8 ピクセル	150	3x3	1.7
	150	5x5	1.71

## ステレオ視差推定用のセミグローバル法 (SemiGlobalBM)

ステレオ マッチング アルゴリズムは、補正されたステレオ画像のペアから相対深度を見つけるために使用されます。結果の視差情報は、ステレオ カメラの既知の内部および外部パラメーターを使用した三角測量による 3 次元再構成に使用できます。ステレオ視差推定用のセミグローバル法は、複数のパス間の差に関するコストを集約し、視差マップのより平滑な推定が得られます。

xfOpenCV のセミグローバル法では、センサ変換をハミング距離と共に使用し、コスト計算を実行します。セミグローバル最適化ブロックはHirschmuller による実装に基づいていますが、4 方向のみを考慮することによりコスト集約を近似しています。

複数視差のコストの計算および集約を並列実行することにより並列処理を達成し、このパラメーターをコンパイル時の入力として含めます。

### API 構文

```
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)
```

### パラメーターの説明

次の表に、テンプレートと関数のパラメーターを説明します。

表 402: SemiGlobalBM 関数のパラメーターの説明

パラメーター	説明
BORDER_TYPE	境界ピクセルをセンサ変換関数で処理する際に使用されるパラメーター。XF_BORDER_CONSTANT のみサポート。
WINDOW_SIZE	センサ変換の計算に使用されるウィンドウのサイズ。5 (5x5) のみサポート。
NDISP	視差の数
PU	並列計算する視差ユニットの数
R	コスト集約に使用する方向の数。2、3、または 4。
SRC_T	入力画像 Mat オブジェクトのタイプ。XF_8UC1 のみサポート。
DST_T	出力視差画像 Mat オブジェクトのタイプ。XF_8UC1 のみサポート。
ROWS	入力画像の最大高さ。
COLS	入力画像の最大幅。
NPC	並列計算されるピクセルの数。XF_NPPC1 のみサポート。
_src_mat_l	左入力画像 Mat
_src_mat_r	右入力画像 Mat
_dst_mat	出力視差画像 Mat
p1	コスト集約の小さいペナルティ
p2	コスト集約の大きいペナルティ。最大値は 100。



## リソース使用量

次の表に、64 個の視差と 32 個の並列ユニットを含む 1920 x 1080 画像でのリソース使用量を示します。

表 403: SemiGlobalBM 関数のリソース使用量のサマリ

動作モード	フィルター サイズ	動作周波数 (MHz)	リソース使用量			
			BRAM_18k	DSP48E	FF	LUT
1 ピクセル	5x5	200	205	141	11856	19102

## パフォーマンス見積もり

次の表に、1920x1080 画像でのパフォーマンス見積もりを示します。

表 404: SemiGlobalBM 関数のパフォーマンス見積もりのサマリ

動作モード	動作周波数	視差の数	並列ユニット	レイテンシ
1 ピクセル/クロック	200 MHz	64	32	42 ms

## ステレオ ローカル ブロック マッチング (StereoBM)

ステレオ ブロック マッチングは、連続するフレーム (ステレオ ペア) 間のブロックの動きを推定します。この前提となるのは、ステレオ ペアでは、前景の物体の方が背景の物体よりも視差が大きいということです。ローカル ブロック マッチングでは、ウィンドウサイズに基づく近傍パッチの情報を使用して、ステレオ ペアの共役点を特定します。グローバル マッチングでは、マッチング ピクセルを計算するのに画像全体からの情報が使用されるので、ローカル マッチングよりも高い精度が得られますが、グローバル マッチングではリソースが多く使用されるので、その点ではローカル マッチングの方が有利です。

ローカル ブロック マッチング アルゴリズムには、前処理処理段階と視差推定段階があります。前処理では、Sobel 勾配計算が実行された後、画像クリッピングが実行されます。視差推定では、SAD (絶対差の和) 計算を実行し、WTA (Winner Takes All) 方式を使用して視差を取得します (最小の SAD が視差)。ほかの可能な視差と異なる場合、ピクセルは無効になります。無効なピクセルは、視差値 0 で示されます。

## API 構文

```
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);
```

## パラメーターの説明

次の表に、テンプレートと関数のパラメーターを説明します。

表 405: StereoBM 関数のパラメーターの説明

パラメーター	説明
WSIZE	視差計算に使用されるウィンドウのサイズ
NDISP	視差の数

表 405: StereoBM 関数のパラメーターの説明 (続き)

パラメーター	説明
NDISP_UNITS	並列計算される視差の数。
SRC_T	入力ピクセルのデータ型。8 ビット、符号なし、1 チャンネル (XF_8UC1) のみサポート。
DST_T	出力タイプ。XF_16UC1 で、視差が Q12.4 フォーマットで配列されます。
ROWS	入力および出力画像の最大高さ
COLS	入力および出力画像の最大幅
NPC	1 サイクルごとに処理されるピクセル数。可能なオプションは XF_NPPC1 のみ。
USE_URAM	一部のストレージ構造を UltraRAM にマップ
left_image	左カメラからの画像
right_image	右カメラからの画像
disparity_image	画像として出力された視差。
sbmstate	ステレオ ブロック マッチング アルゴリズムに関するさまざまなパラメーターで構成されるクラス オブジェクト。 1. preFilterCap: デフォルト値は 31 で、有効な値は 1 ~ 63 です。 2. minDisparity: デフォルト値は 0 で、有効な値は 0 ~ (imgWidth-NDISP) です。 3. uniquenessRatio: デフォルト値は 15 で、有効な値は負でない整数です。 4. textureThreshold: デフォルト値は 10 で、有効な値は負でない整数です。

## リソース使用量

次の表に、ザイリンクス Xczu9eg-ffvb1156-1-i-es1 FPGA でグレースケール HD (1080x1920) 画像を処理するために、Vivado HLS 2019.1 ツールを使用して生成された異なる設定でのカーネルのリソース使用量を示します。

設定のフォーマットは imageSize\_WSIZE\_NDisp\_NDispUnits です。

表 406: StereoBM 関数のリソース使用量のサマリ

設定	周波数 (MHz)	リソース使用量			
		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

次の表に、ザイリンクス xczu7ev-ffvc1156-2-e FPGA で UltraRAM をイネーブルにしてグレースケール HD (1080x1920) 画像を処理するために、SDx 2019.1 ツールを使用して生成された異なる設定でのカーネルのリソース使用量を示します。

設定のフォーマットは imageSize\_WSIZE\_NDisp\_NDispUnits です。

表 407: UltraRAM をイネーブルにした場合の StereoBM 関数のリソース使用量のサマリ

設定	周波数 (MHz)	リソース使用量				
		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

### パフォーマンス見積もり

次の表に、ザイリンクス Xczu9eg-ffvb1156-1-i-es1 FPGA でグレースケール HD (1080x1920) 画像を処理するために、Vivado HLS 2019.1 ツールを使用して生成された異なる設定でのステレオ ローカル ブロック マッチングのパフォーマンス見積もりを示します。

設定のフォーマットは imageSize\_WSIZE\_NDisp\_NDispUnits です。

表 408: StereoBM 関数のパフォーマンス見積もりのサマリ

設定	周波数 (MHz)	レイテンシ (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

subRS 関数は、スカラー画像からソース画像の強度を減算し、その結果をデスティネーション画像に保存します。

$$\text{dst}(l) = \text{scl} - \text{src}(l)$$

### API 構文

```
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
_src1[XF_CHANNELS(SRC_T, NPC)], xf::Mat<SRC_T, ROWS, COLS, NPC> & _dst)
```

### パラメーターの説明

次の表に、テンプレートと関数のパラメーターを説明します。

表 409: subRS 関数のパラメーターの説明

パラメーター	説明
SRC_T	入力ピクセルのデータ型。8 ビット、符号なし、1 チャンネル (XF_8UC1) をサポート。
ROWS	入力および出力画像の最大高さ
COLS	入力および出力画像の最大幅。N ピクセルの並列処理の場合、幅は N の倍数で指定。
NPC	1 サイクルごとに処理されるピクセル数。1 ピクセルの場合は XF_NPPC1、8 ピクセルの場合は XF_NPPC8。
_src1	1 つ目の入力画像
_scl	入力スカラー値。サイズはチャンネル数にする必要があります。
_dst	出力画像

### リソース使用量

次の表に、Xczu9eg-ffvb1156-1-i-es1 FPGA で Vivado HLS 2019.1 バージョン ツールを使用して生成された、リソース最適化 (8 ピクセル) モードおよび通常モードにおける subRS 関数のリソース使用量を示します。

表 410: subRS 関数のリソース使用量のサマリ

名前	リソース使用量	
	1 クロックごとに 1 ピクセル動作	8 クロックごとに 1 ピクセル動作
	300 MHz	150 MHz
BRAM_18K	0	0
DSP48E	0	0
FF	103	104
LUT	44	133
CLB	23	43

### パフォーマンス見積もり

次の表に、Xczu9eg-ffvb1156-1-i-es1 FPGA でグレースケール HD (1080x1920) 画像を処理するために、Vivado HLS 2019.1 ツールを使用して生成された異なる設定でのカーネルのパフォーマンス見積もりを示します。

表 411: subRS 関数のパフォーマンス見積もりのサマリ

動作モード	レイテンシ見積もり	
	動作周波数 (MHz)	レイテンシ (ms)
1 ピクセル	300	6.9
8 ピクセル	150	1.7

## SubS

subS 関数は、ソース画像の強度からスカラー値を減算し、その結果をデスティネーション画像に保存します。

$$\text{dst}(l) = \text{src}(l) - \text{scl}$$

## API 構文

```
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)
```

## パラメーターの説明

次の表に、テンプレートと関数のパラメーターを説明します。

表 412: subS 関数のパラメーターの説明

パラメーター	説明
SRC_T	入力ピクセルのデータ型。8 ビット、符号なし、1 チャンネル (XF_8UC1) をサポート。
ROWS	入力および出力画像の最大高さ。
COLS	入力および出力画像の最大幅。N ピクセルの並列処理の場合、幅は N の倍数で指定。
NPC	1 サイクルごとに処理されるピクセル数。1 ピクセルの場合は XF_NPPC1、8 ピクセルの場合は XF_NPPC8。
_src1	1 つ目の入力画像
_scl	入力スカラー値。サイズはチャンネル数にする必要があります。
_dst	出力画像

## リソース使用量

次の表に、Xczu9eg-ffvb1156-1-i-es1 FPGA で Vivado HLS 2019.1 バージョンツールを使用して生成された、リソース最適化 (8 ピクセル) モードおよび通常モードにおける subS 関数のリソース使用量を示します。

表 413: subS 関数のリソース使用量のサマリ

名前	リソース使用量	
	1 クロックごとに 1 ピクセル動作	8 クロックごとに 1 ピクセル動作
	300 MHz	150 MHz
BRAM_18K	0	0
DSP48E	0	0
FF	103	104
LUT	44	133
CLB	23	43

## パフォーマンス見積もり

次の表に、Xczu9eg-ffvb1156-1-i-es1 FPGA でグレースケール HD (1080x1920) 画像を処理するために、Vivado HLS 2019.1 ツールを使用して生成された異なる設定でのカーネルのパフォーマンス見積もりを示します。

表 414: subS 関数のパフォーマンス見積もりのサマリ

動作モード	レイテンシ見積もり	
	動作周波数 (MHz)	レイテンシ (ms)
1 ピクセル	300	6.9

表 414: subS 関数のパフォーマンス見積もりのサマリ (続き)

動作モード	レイテンシ見積もり	
	動作周波数 (MHz)	レイテンシ (ms)
8 ピクセル	150	1.7

## 和 (sum)

sum 関数は、入力画像からすべてのピクセルの和を計算します。

### API 構文

```
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) ])
```

### パラメーターの説明

次の表に、テンプレートと関数のパラメーターを説明します。

表 415: sum 関数のパラメーターの説明

パラメーター	説明
SRC_T	入力ピクセルのデータ型。8 ビット、符号なし、1 チャンネル (XF_8UC1) をサポート。
ROWS	入力および出力画像の最大高さ。
COLS	入力および出力画像の最大幅 (8 の倍数で指定)。
NPC	1 サイクルごとに処理されるピクセル数。
_src1	入力画像。
sum	画像のすべてのピクセルの和を格納する配列。

### リソース使用量

次の表に、Xczu9eg-ffvb1156-1-i-es1 FPGA で Vivado HLS 2019.1 バージョンツールを使用して生成された、リソース最適化 (8 ピクセル) モードおよび通常モードにおける sum 関数のリソース使用量を示します。

表 416: sum 関数のリソース使用量のサマリ

名前	リソース使用量	
	1 クロックごとに 1 ピクセル動作	8 クロックごとに 1 ピクセル動作
	300 MHz	150 MHz
BRAM_18K	0	0
DSP48E	0	0
FF	341	408
LUT	304	338
CLB	71	87

## パフォーマンス見積もり

次の表に、Xczu9eg-ffvb1156-1-i-es1 FPGA でグレースケール HD (1080x1920) 画像を処理するために、Vivado HLS 2019.1 ツールを使用して生成された異なる設定でのカーネルのパフォーマンス見積もりを示します。

表 417: sum 関数のパフォーマンス見積もりのサマリ

動作モード	レイテンシ見積もり	
	動作周波数 (MHz)	レイテンシ (ms)
1 ピクセル	300	
8 ピクセル	150	

## SVM

SVM 関数は SVM コアの演算で、入力配列間のドット積を実行します。この関数は、固定小数点型のドット積値の結果を返します。

### API 構文

```
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)
```

### パラメーターの説明

次の表に、テンプレートと関数のパラメーターを説明します。

表 418: SVM 関数のパラメーターの説明

パラメーター	説明
SRC1_T	入力ピクセルのデータ型。16 ビット、符号付き、1 チャンネル (XF_16SC1) をサポート。
SRC2_T	入力ピクセルのデータ型。16 ビット、符号付き、1 チャンネル (XF_16SC1) をサポート。
DST_T	出力データ型。32 ビット、符号付き、1 チャンネル (XF_32SC1) をサポート。
ROWS1	処理される最初の画像の行数。
COLS1	処理される最初の画像の列数。
ROWS2	処理される 2 つ目の画像の行数。
COLS2	処理される 2 つ目の画像の列数。
NPC	1 サイクルごとに処理されるピクセル数。可能なオプションは XF_NPPC1。
N	カーネル処理の最大数。
in_1	最初の入力配列。
in_2	2 つ目の入力配列。
idx1	最初の配列のインデックスの開始。
idx2	2 つ目の配列のインデックスの開始。
frac1	最初の配列データの小数ビット数。
frac2	2 つ目の配列データの小数ビット数。

表 418: SVM 関数のパラメーターの説明 (続き)

パラメーター	説明
n	カーネル処理の数。
out_frac	結果の値の小数ビット数。
result	結果の値。

## リソース使用量

次の表に、ザイリンクス Xczu9eg-ffvb1156-1-i-es1 FPGA 用に Vivado HLS 2019.1 ツールを使用して生成された SVM 関数のリソース使用量を示します。

表 419: SVM 関数のリソース使用量のサマリ

動作周波数 (MHz)	使用量の見積もり (ms)				
	BRAM_18K	DSP_48E	FF	LUT	CLB
300	0	1	27	34	12

## パフォーマンス見積もり

次の表に、ザイリンクス Xczu9eg-ffvb1156-1-i-es1 FPGA 用に Vivado HLS 2019.1 ツールを使用して生成された異なる設定でのパフォーマンス見積もりを示します。

表 420: SVM 関数のパフォーマンス見積もりのサマリ

動作周波数 (MHz)	レイテンシ見積もり	
	最小 (サイクル数)	最大 (サイクル数)
300	204	204

## しきい値処理 (Threshold)

Threshold 関数は、入力画像のしきい値処理を実行する関数です。この関数では、複数のタイプのしきい値処理がサポートされます。

$$dst(x, y) = \begin{cases} maxval, & \text{if } src(x, y) > threshold \\ 0, & \text{Otherwise} \end{cases}$$

$$dst(x, y) = \begin{cases} 0, & \text{if } src(x, y) > threshold \\ maxval, & \text{Otherwise} \end{cases}$$

$$dst(x, y) = \begin{cases} threshold, & \text{if } src(x, y) > threshold \\ src(x, y), & \text{Otherwise} \end{cases}$$

$$dst(x, y) = \begin{cases} src(x, y), & \text{if } src(x, y) > threshold \\ 0, & \text{Otherwise} \end{cases}$$



$$dst(x, y) = \begin{cases} 0, & \text{if } src(x, y) > threshold \\ src(x, y), & \text{Otherwise} \end{cases}$$

## API 構文

```
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 )
```

## パラメーターの説明

次の表に、テンプレートと関数のパラメーターを説明します。

表 421: Threshold 関数のパラメーターの説明

パラメーター	説明
THRESHOLD_TYPE	しきい値処理のタイプ。
SRC_T	入力ピクセルのデータ型。8 ビット、符号なし、1 チャンネル (XF_8UC1) のみサポート。
ROWS	入力および出力画像の最大高さ。
COLS	入力および出力画像の最大幅。8 ピクセル動作の場合は 8 の倍数で指定。
NPC	1 サイクルごとに処理されるピクセル数。
_src_mat	入力画像
_dst_mat	出力画像
thresh	しきい値。
maxval	THRESH_BINARY および THRESH_BINARY_INV しきい値処理タイプに使用される最大値。

## リソース使用量

次の表に、ザイリンクス Xczu9eg-ffvb1156-1 FPGA でグレースケール HD (1080x1920) 画像を処理するために、Vivado HLS 2019.1 ツールを使用して生成された異なる設定でのバイナリしきい値処理を使用したカーネルのリソース使用量を示します。

表 422: Threshold 関数のリソース使用量のサマリ

設定	リソース使用量	
	1 ピクセル	8 ピクセル
	300 MHz	150 MHz
BRAM_18K	0	0
DSP48E	0	0
FF	110	154
LUT	61	139
CLB	16	37

## パフォーマンス見積もり

次の表に、ザイリンクス Xczu9eg-ffvb1156-1 でグレースケール HD (1080x1920) 画像を処理するために、Vivado HLS 2019.1 ツールを使用して生成された異なる設定でのカーネルのパフォーマンス見積もりを示します。

表 423: Threshold 関数のパフォーマンス見積もりのサマリ

動作モード	動作周波数 (MHz)	レイテンシ見積もり (ms)
1 ピクセル	300	7.2
8 ピクセル	150	1.7

## Atan2

Atan2LookupFP 関数は、 $y/x$  の逆正接関数を計算します。ベクター  $\begin{bmatrix} x \\ y \end{bmatrix}$  の原点に対する角度を返します。atan2 により返される角度には、象限情報も含まれます。

Atan2LookupFP 関数は、標準の atan2 関数の固定小数点バージョンです。この関数は、ルックアップテーブル方法を使用して atan2 をインプリメントします。ルックアップテーブルの値は Q4.12 フォーマットで表現されるので、返される値も Q4.12 フォーマットです。glibc に含まれる atan2 関数と比較すると、89°～90°の範囲で 0.2°の最大誤差があります。それ以外の角度 (0°～89°) の最大誤差は 10-3 程度です。xs と ys の両方が 0 の場合は、0 が返されます。

## API 構文

```
short Atan2LookupFP(short xs, short ys, int M1,int N1,int M2, int N2)
```

## パラメーターの説明

次の表に、テンプレートと関数のパラメーターを説明します。

表 424: Atan2LookupFP 関数のパラメーターの説明

パラメーター	説明
xs	QM1.N1 の固定小数点フォーマットの 16 ビット符号なし値 x
ys	QM2.N2 の固定小数点フォーマットの 16 ビット符号なし値 y
M1	x の整数部分を表すビット数。
N1	y の分数部分を表すビット数。16-M1 にする必要があります。
M2	y の整数部分を表すビット数。
N2	y の分数部分を表すビット数。16-N1 にする必要があります。
Return	戻り値 (ラジアン)。Q4.12 の固定小数点フォーマットで $-\pi \sim +\pi$ の範囲です。

## リソース使用量

次の表に、ザイリンクス Xczu9eg-ffvb1156-1-i-es1 FPGA 用に Vivado HLS 2019.1 ツールを使用して生成された Atan2LookupFP 関数のリソース使用量を示します。

表 425: Atan2LookupFP 関数のリソース使用量のサマリ

動作周波数 (MHz)	使用量の見積もり				
	BRAM_18K	DSP_48E	FF	LUT	CLB
300	4	2	275	75	139

### パフォーマンス見積もり

次の表に、サイリンクス Xczu9eg-ffvb1156-1-i-es1 FPGA 用に Vivado HLS 2019.1 ツールを使用して生成された異なる設定でのパフォーマンス見積もりを示します。

表 426: Atan2LookupFP 関数のパフォーマンス見積もりのサマリ

動作周波数 (MHz)	レイテンシ見積もり	
	最小 (サイクル数)	最大 (サイクル数)
300	1	15

## インバース/逆数 (Inverse)

Inverse 関数では、数値  $x$  の逆数が計算されます。 $1/x$  値は 2048 サイズのルックアップテーブルに格納されています。 $1/x$  値を選択するインデックスは、 $x$  の固定小数点フォーマットを使用して計算されます。このインデックスが計算されると、ルックアップテーブルから対応する  $1/x$  値がフェッチされ、この値とこの値を固定小数点で表すために必要な小数ビット数が返されます。

### API 構文

```
unsigned int Inverse(unsigned short x,int M,char *N)
```

### パラメーターの説明

次の表に、テンプレートと関数のパラメーターを説明します。

表 427: Inverse 関数のパラメーターの説明

パラメーター	説明
x	QM の固定小数点フォーマットの 16 ビット符号なし値 $x$ (16-M)
M	$x$ の整数部分を表すビット数。
N	$1/x$ の小数部分を表すためのビット数を格納する char 変数へのポインター。この値が関数から返されます。
Return	$Q(32-N).N$ の固定小数点フォーマットで表された 32 ビットフォーマットで返される $1/x$ 値

### リソース使用量

次の表に、サイリンクス Xczu9eg-ffvb1156-1-i-es1 FPGA 用に Vivado HLS 2019.1 ツールを使用して生成された Inverse 関数のリソース使用量を示します。

表 428: Inverse 関数のリソース使用量のサマリ

動作周波数 (MHz)	使用量の見積もり (ms)				
	BRAM_18K	DSP_48E	FF	LUT	CLB
300	4	0	68	128	22

### パフォーマンス見積もり

次の表に、ザイリンクス Xczu9eg-ffvb1156-1-i-es1 FPGA 用に Vivado HLS 2019.1 ツールを使用して生成された異なる設定でのパフォーマンス見積もりを示します。

表 429: Inverse 関数のパフォーマンス見積もりのサマリ

動作周波数 (MHz)	レイテンシ見積もり	
	最小 (サイクル数)	最大 (サイクル数)
300	1	8

## ルックアップ テーブル (LUT)

LUT 関数では、テーブルルックアップ処理が実行されます。指定したルックアップテーブルを使用してソース画像がデスティネーション画像に変換されます。入力画像は XF\_8UP の深さで、出力画像タイプも入力画像タイプと同じである必要があります。

$$I_{out}(x, y) = LUT [I_{in1}(x, y)]$$

説明:

- $I_{out}(x, y)$ : 出力画像の (x,y) 位置での強度
- $I_{in}(x, y)$ : 最初の入力画像の (x,y) 位置での強度
- LUT: サイズ 256 および符号なしの char 型のルックアップ テーブル。

### API 構文

```
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)
```

### パラメーターの説明

次の表に、テンプレートと関数のパラメーターを説明します。

表 430: LUT 関数のパラメーターの説明

パラメーター	説明
SRC_T	入力および出力のピクセルのデータ型。8ビット、符号なし、1 および 3 チャンネル (XF_8UC1、XF_8UC3) のみサポート。
ROWS	処理される画像の行数。

表 430: LUT 関数のパラメーターの説明 (続き)

パラメーター	説明
COLS	処理される画像の列数。8 ピクセル動作の場合は 8 の倍数で指定。
NPC	並列で処理されるピクセル数。使用可能なオプションは、1 ピクセルの場合は XF_NPPC1、8 ピクセルの場合は XF_NPPC8。
_src	サイズ (ROWS, COLS) および 8U 型の入力画像。
_dst	サイズ (ROWS, COLS) で入力と同型の出力画像。
_lut	サイズ 256 および符号なしの char 型のルックアップ テーブル。

## リソース使用量

次の表に、ザイリンクス Xczu9eg-ffvb1156-1-i-es1 FPGA でグレースケール HD (1080x1920) 画像を処理するために、Vivado HLS 2019.1 ツールを使用して生成された LUT 関数のリソース使用量を示します。

表 431: LUT 関数のリソース使用量のサマリ

動作モード	動作周波数 (MHz)	使用量の見積もり				
		BRAM_18K	DSP_48E	FF	LUT	CLB
1 ピクセル	300	1	0	937	565	137
8 ピクセル	150	9	0	1109	679	162

次の表に、ザイリンクス Xczu9eg-ffvb1156-1-i-es1 FPGA で 3 チャンネルの 4K 画像を処理するために、Vivado HLS 2019.1 ツールを使用して生成された LUT 関数のリソース使用量を示します。

表 432: LUT 関数のリソース使用量のサマリ

動作モード	動作周波数 (MHz)	使用量の見積もり				
		BRAM_18K	DSP_48E	FF	LUT	CLB
1 ピクセル	300	4	0	1160	648	175

## パフォーマンス見積もり

次の表に、ザイリンクス Xczu9eg-ffvb1156-1-i-es1 でグレースケール HD (1080x1920) 画像を処理するために、Vivado HLS 2019.1 ツールで生成された異なる設定でのパフォーマンス見積もりを示します。

表 433: LUT 関数のパフォーマンス見積もりのサマリ

動作モード	レイテンシ見積もり
	最大レイテンシ
1 ピクセル動作 (300 MHz)	6.92 ms
8 ピクセル動作 (150 MHz)	1.66 ms

## 平方根 (Sqrt)

Sqrt 関数では、引き放し法の平方根アルゴリズムを使用して 16 ビットの固定小数点の平方根が計算されます。引き放し法の平方根アルゴリズムでは、平方根結果に 2 の補数表記が使用されます。このアルゴリズムでは、イテレーションごとに最後のビットでも正確な結果値を生成できます。

入力引数 D は、32 ビットと宣言されていますが、16 ビットにする必要があります。出力 sqrt(D) は 16 ビット型です。D のフォーマットが QM.N (M+N = 16) の場合、出力フォーマットは Q(M/2).N です。

小数部の n ビットの精度は、関数呼び出し前に被開数 (D) を左に 2n シフトし、その解を右に n シフトすると取得できます。たとえば、35 (01100011<sub>2</sub>) の平方根を小数点後のビット数 1 (N=1) で求める方法は次のとおりです。

1. 値 (0110001100<sub>2</sub>) を左に 2 シフト。
2. 得られた値 (1011<sub>2</sub>) を右に 1 シフト。正しい答えは 101.1 (5.5) となります。

## API 構文

```
int Sqrt(unsigned int D)
```

## パラメーターの説明

次の表に、テンプレートと関数のパラメーターを説明します。

表 434: Sqrt 関数のパラメーターの説明

パラメーター	説明
D	16 ビット固定小数点フォーマットの入力データ。
Return	short int フォーマットの出力値。

## リソース使用量

次の表に、サイリンクス Xczu9eg-ffvb1156-1-i-es1 FPGA 用に Vivado HLS 2019.1 ツールを使用して生成された Sqrt 関数のリソース使用量を示します。

表 435: Sqrt 関数のリソース使用量のサマリ

動作周波数 (MHz)	使用量の見積もり				
	BRAM_18K	DSP_48E	FF	LUT	CLB
300	0	0	8	6	1

## パフォーマンス見積もり

次の表に、サイリンクス Xczu9eg-ffvb1156-1-i-es1 FPGA 用に Vivado HLS 2019.1 ツールを使用して生成された異なる設定でのパフォーマンス見積もりを示します。

表 436: Sqrt 関数のパフォーマンス見積もりのサマリ

動作周波数 (MHz)	レイテンシ見積もり	
	最小 (サイクル数)	最大 (サイクル数)
300	18	18

## ワープ変換 (warpTransform)

`warpTransform` 関数は、画像に対して透視変換およびアフィン幾何変換を実行します。変換のタイプは、関数に対するコンパイル時パラメーターです。

この関数では、ストリーミング インターフェイスを使用して変換を実行します。このことと、幾何変換で 1 つの出力行を計算するのに入力データの多数の行にアクセスする必要があるということから、入力データの一部の行がブロック RAM/UltraRAM に保存されます。保存される行数は、テンプレート パラメーターを使用して設定できます。変換行列に基づいて、保存する行数を決定できます。また、入力画像の変換をいつ開始するかを保存される画像の行数を単位として指定できます。

### アフィン変換

変換行列は、次に示すように、サイズ パラメーターで構成されます。

$$M = \begin{bmatrix} M_{11} & M_{12} & M_{13} \\ M_{21} & M_{22} & M_{23} \end{bmatrix}$$

`warpTransform` 関数では、アフィン変換は次の式に従って適用されます。

$$dst \begin{pmatrix} x \\ y \end{pmatrix} = M * src \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}$$

### 透視変換

変換行列は、次に示すように 3x3 行列です。

$$M = \begin{bmatrix} M_{11} & M_{12} & M_{13} \\ M_{21} & M_{22} & M_{23} \\ M_{31} & M_{32} & M_{33} \end{bmatrix}$$

`warpTransform` 関数では、透視変換は次の式に従って適用されます。

$$dst^1 \begin{pmatrix} x \\ y \\ n \end{pmatrix} = M * src \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}$$

その後、`dst1` の最初の 2 つの次元を 3 つ目の次元で割って変換後のピクセルが計算されます。

$$dst^1 \begin{pmatrix} x \\ y \\ n \end{pmatrix} = M * src \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}$$

## API 構文

```
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)
```

## パラメーターの説明

次の表に、テンプレートと関数のパラメーターを説明します。

表 437: warpTransform 関数のパラメーターの説明

パラメーター	説明
STORE_LINES	指定した変換を実行するために入力を格納する行数。
START_ROW	画像の変換を開始する前に保存する入力行の数。STORE_LINES の値以下にする必要があります。
TRANSFORMATION_TYPE	アフィン変換および透視変換がサポートされます。0 に設定するとアフィン変換、1 に設定すると透視変換が実行されます。
INTERPOLATION_TYPE	1 に設定するとバイリニア補間、0 に設定すると最近傍補間が使用されます。
SRC_T	入力および出力のピクセルのデータ型。8 ビット、符号なし、1 および 3 チャンネル (XF_8UC1、XF_8UC3) のみサポート。
ROWS	入力および出力画像の最大高さ。
COLS	入力および出力画像の最大幅。
NPC	1 サイクルごとに処理されるピクセル数。可能なオプションは 1 ピクセル操作 (XF_NPPC1) のみ。
USE_URAM	一部のストレージ構造を UltraRAM にマップ
src	入力画像
dst	出力画像
transformation_matrix	入力画像に適用する変換行列。

## リソース使用量

次の表に、ザイリンクス Xczu9eg-ffvb1156-1-i-es1 FPGA でローカルバッファに画像の行数を格納してグレースケール HD (1080x1920) 画像を処理するために、Vivado HLS 2019.1 ツールを使用して生成されたワープ変換のリソース使用量を示します。

表 438: warpTransform 関数のリソース使用量のサマリ

変換	INTERPOLATION_TYPE	STORE_LINES	START_ROW	動作周波数 (MHz)	使用量の見積もり			
					LUT	FF	DSP	BRAM
透視変換	バイリニア	100	50	300	7468	9804	61	112
透視変換	最近傍	100	50	300	4514	6761	35	104
アフィン変換	バイリニア	100	50	300	6139	5606	40	124
アフィン変換	最近傍	100	50	300	4611	4589	18	112

次の表に、ザイリンクス Xczu9eg-ffvb1156-1-i-es1 FPGA でローカルバッファに画像の行数を格納して BGR 4K 画像を処理するために、Vivado HLS 2019.1 ツールを使用して生成されたワープ変換のリソース使用量を示します。



表 439: warpTransform 関数のリソース使用量のサマリ

変換	INTERPOLATION _TYPE	STORE _LINES	START _ROW	動作周波数 (MHz)	使用量の見積もり			
					LUT	FF	DSP	BRAM
透視変換	バイリニア	100	50	300	9192	7910	48	616
透視変換	最近傍	100	50	300	10533	12055	69	604
アフィン変換	バイリニア	100	50	300	6397	8415	35	604

次の表に、ザイリンクス xczu7ev-ffvc1156-2-e FPGA で UltraRAM をイネーブルにしてグレースケール 4K 画像を処理するために、SDx 2019.1 ツールを使用して生成されたワープ変換のリソース使用量を示します。

表 440: UltraRAM をイネーブルにした場合の warpTransform 関数のリソース使用量のサマリ

変換	INTERPOLATION _TYPE	STORE _LINES	START _ROW	動作周波数 (MHz)	使用量の見積もり				
					LUT	FF	DSP	BRAM	URAM
透視変換	バイリニア	100	50	300	7820	12458	61	7	12
透視変換	最近傍	100	50	300	4880	8323	35	2	6
アフィン変換	バイリニア	100	50	300	6850	9516	40	13	12
アフィン変換	最近傍	100	50	300	4651	6548	18	6	6

## パフォーマンス見積もり

次の表に、ザイリンクス Xczu9eg-ffvb1156-1-i-es1 FPGA でグレースケール HD (1080x1920) 画像を処理するために、Vivado HLS 2019.1 ツールを使用して生成されたワープ変換のパフォーマンス見積もりを示します。

表 441: warpTransform 関数のパフォーマンス見積もりのサマリ

変換	INTERPOLATION _TYPE	STORE _LINES	START _ROW	動作周波数 (MHz)	レイテンシ見積もり 最大 (ms)
透視変換	バイリニア	100	50	300	7.46
透視変換	最近傍	100	50	300	7.31
アフィン変換	バイリニア	100	50	300	7.31
アフィン変換	最近傍	100	50	300	7.24

## zero

zero 関数は、入力画像の各ピクセルを 0 に設定し、結果を dst に保存します。

## API 構文

```
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)
```

## パラメーターの説明

次の表に、テンプレートと関数のパラメーターを説明します。

表 442: zero 関数のパラメーターの説明

パラメーター	説明
SRC_T	入力ピクセルのデータ型。8 ビット、符号なし、1 チャンネル (XF_8UC1) をサポート。
ROWS	入力および出力画像の最大高さ。
COLS	入力および出力画像の最大幅。N ピクセルの並列処理の場合、幅は N の倍数で指定。
NPC	1 サイクルごとに処理されるピクセル数。1 ピクセルの場合は XF_NPPC1、8 ピクセルの場合は XF_NPPC8。
_src1	入力画像
_dst	出力画像

## リソース使用量

次の表に、Xczu9eg-ffvb1156-1-i-es1 FPGA で Vivado HLS 2019.1 バージョン ツールを使用して生成された、リソース最適化 (8 ピクセル) モードおよび通常モードにおける zero 関数のリソース使用量を示します。

表 443: zero 関数のリソース使用量のサマリ

名前	リソース使用量	
	1 クロックごとに 1 ピクセル動作	8 クロックごとに 1 ピクセル動作
	300 MHz	150 MHz
BRAM_18K	0	0
DSP48E	0	0
FF	78	78
LUT	42	41
CLB	15	14

## パフォーマンス見積もり

次の表に、Xczu9eg-ffvb1156-1-i-es1 FPGA でグレースケール HD (1080x1920) 画像を処理するために、Vivado HLS 2019.1 ツールを使用して生成された異なる設定でのカーネルのパフォーマンス見積もりを示します。

表 444: zero 関数のパフォーマンス見積もりのサマリ

動作モード	レイテンシ見積もり	
	動作周波数 (MHz)	レイテンシ (ms)
1 ピクセル	300	6.9
8 ピクセル	150	1.7

# xfOpenCV ライブラリを使用したデザイン例

ライブラリに含まれるすべてのハードウェア関数には、それに対応するデザイン例が GitHub から提供されています。このセクションでは、xfOpenCV に含まれるさまざまな関数を組み合わせてインプリメントした画像処理関数とパイプラインの詳細を説明します。プロセッサとプログラマブル ロジックの両方の機能を利用してさまざまな機能をインプリメントするのに最適な方法を示します。これらの例では、複雑なデータフローパスをインプリメントする異なる方法も示します。このセクションでは、次の例について説明します。

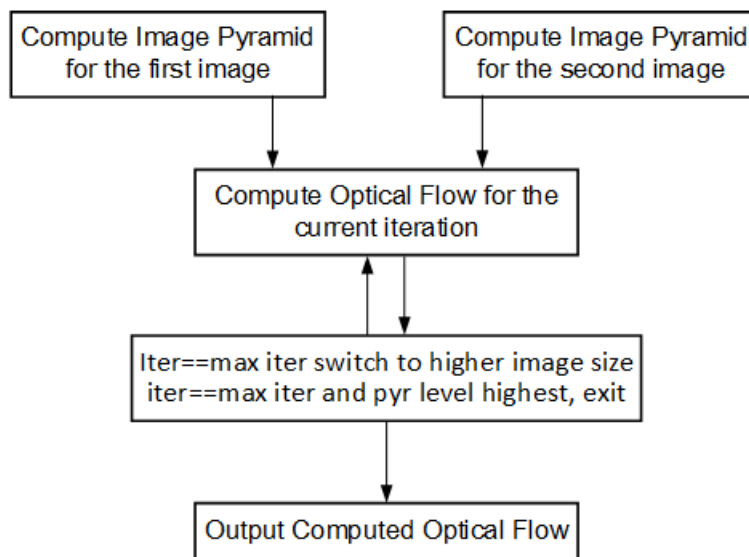
- 反復ピラミッド型高密度オプティカル フロー
- 疎なオプティカル フローを使用したコーナー追跡
- 色検出
- ガウシアン フィルターの差
- ステレオ ビジョン パイプライン

---

## 反復ピラミッド型高密度オプティカル フロー

高密度ピラミッド型オプティカル フローの例は、xfOpenCV ライブラリの `xf::pyrDown` および `xf::densePyrOpticalFlow` ハードウェア関数を使用して画像ピラミッドを作成し、それに対して反復実行して、2つの入力画像間のオプティカル フローを計算します。この例では、`xf::pyrDown` 関数のハードウェア インスタンスを 2つ使用して、2つの入力画像の画像ピラミッドを並列に計算します。2つの画像ピラミッドは、`xf::densePyrOpticalFlow` 関数の 1つのハードウェア インスタンスにより、最小画像サイズから最大画像サイズまで処理されます。各反復の出力フロー ベクターは、ハードウェア カーネルにハードウェア関数への入力としてフィードバックされます。最大画像サイズの最後の反復の出力が、高密度ピラミッド型オプティカル フローの出力となります。

図 11: 反復ピラミッド型高密度オプティカル フロー



次に、ホストにおけるこの例のインプリメンテーションの詳細を説明します。これらの詳細は、必要なスループットを達成するためのプロセスを理解するのに役立ちます。

## pyrof\_hw()

pyrof\_hw() は、高密度オプティカル フローを計算するホスト関数です。

### API 構文

```
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])
```

### パラメーターの説明

次の表に、テンプレートと関数のパラメーターを説明します。

パラメーター	説明
im0	cv::Mat の 1 つ目の入力画像
im1	cv::Mat の 2 つ目の入力画像
flowUmat	出力フロー ベクターの水平要素を保存するために割り当てられた cv::Mat
flowVmat	出力フロー ベクターの垂直要素を保存するために割り当てられた cv::Mat
flow	ハードウェア関数を使用した反復計算中に、バックされたフロー ベクターを一時的に格納するために割り当てられた xf::Mat
flow_iter	ハードウェア関数を使用した反復計算中に、バックされたフロー ベクターを一時的に格納するために割り当てられた xf::Mat
mat_imagepyr1	xf::Mat のサイズがピラミッド レベル数に等しい配列。最初の画像のピラミッド レベル数を保存します。

パラメーター	説明
mat_imagepyr2	xf::Mat のサイズがピラミッド レベル数に等しい配列。2 つ目の画像のピラミッド レベル数を保存します。
pyr_h	各ピラミッド レベルで画像の高さを格納するための、画像のピラミッド レベル数を含む整数の配列
pyr_w	各ピラミッド レベルで画像の幅を格納するための、画像のピラミッド レベル数を含む整数の配列

## データフロー

pyroflow\_hw() 関数は、次を実行します。

1. 画像ピラミッドのさまざまなレベルの画像サイズを設定
2. 入力画像を cv::Mat フォーマットから最大画像ピラミッド レベルを含めるよう割り当てられている xf::Mat オブジェクトにコピー
3. pyr\_dense\_optical\_flow\_pyr\_down\_accel() 関数を呼び出して画像ピラミッドを作成
4. pyr\_dense\_optical\_flow\_accel() 関数を使用して、ユーザーが入力したピラミッド レベルすべてで反復してオプティカル フロー出力を計算
5. フロー ベクターをアンパックして浮動小数点に変換し、返す

上記の重要な手順 3 および 4 について、次に詳細に説明します。

## pyr\_dense\_optical\_flow\_pyr\_down\_accel()

### API 構文

```
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])
```

### パラメーターの説明

次の表に、テンプレートと関数のパラメーターを説明します。

パラメーター	説明
mat_imagepyr1	xf::Mat のサイズがピラミッド レベル数に等しい配列。最初の画像のピラミッド レベル数を保存します。この割り当てられたメモリの最上位ピラミッド レベル [0] に対応するメモリ ロケーションに、最初の入力画像が含まれている必要があります。
mat_imagepyr2	xf::Mat のサイズがピラミッド レベル数に等しい配列。2 つ目の画像のピラミッド レベル数を保存します。この割り当てられたメモリの最上位ピラミッド レベル [0] に対応するメモリ ロケーションに、2 番目の入力画像が含まれている必要があります。

pyr\_dense\_optical\_flow\_pyr\_down\_accel() は、次のように、xf::pyrDown ハードウェア関数を呼び出して 1 つの for ループを実行します。

```
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_comp],
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_comp], mat_imagepyr2[pyr_comp+1]);
#pragma SDS wait(1)
#pragma SDS wait(2)
}
```

プラグマを除けばコードは単純で、各反復で `xf::pyrDown` 関数が 2 回呼び出されます。1 回目は最初の画像に対して、2 回目は 2 つ目の画像に対して実行されます。次の反復への入力、現在の反復の出力です。プラグマ `#pragma SDS async(ID)` は、Arm® プロセッサがハードウェア関数を呼び出し、ハードウェア関数が返されるのを待たないように指示します。Arm プロセッサが関数を呼び出すのに数サイクルかかります。これには、DAM のプログラムが含まれます。プラグマ `#pragma SDS wait(ID)` は、Arm プロセッサが `async(ID)` プラグマで呼び出されたハードウェア関数が処理を完了するのを待つように指示します。プラグマ `#pragma SDS resource(ID)` は、ハードウェア関数が異なる ID で呼び出されるたびに、別のハードウェアインスタンスを作成します。つまり、上記のホスト関数のループは、`xf::pyrDown` 関数のハードウェアインスタンス 2 つを並列に呼び出し、両方の関数が戻るまで待ってから、次の反復に進みます。

### 高密度ピラミッド型オプティカル フローの計算

```
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 optical flow computation
} // end pyramidal iterative optical flow HLS computation
```

反復ピラミッド型オプティカル フローは、入れ子の for ループで計算されます。ループは、反復回数にピラミッド レベル数をかけた回数実行されます。メインのループは最小の画像サイズから開始し、最大の画像サイズまで繰り返されます。1 ピラミッド レベルの反復が実行される前に、現在のピラミッド レベルの高さと幅が `curr_height` および `current_width` 変数に設定されます。入れ子のループでは、スケール アップが必要な場合は `next_height` 変数が最初の反復にある前の画像の高さに設定されます。除算はコストが高く、ハードウェアでは一度だけの除算は回避されるので、ホストでスケール係数が計算され、引数としてハードウェア カーネルに渡されます。各ピラミッド レベルの後、スケール アップ フラグがセットされ、入力フロー ベクターを次に大きい画像サイズにスケール アップする必要があることがハードウェア関数に通知されます。スケール アップは、ハードウェア カーネルのバイリニア補間を使用して実行されます。

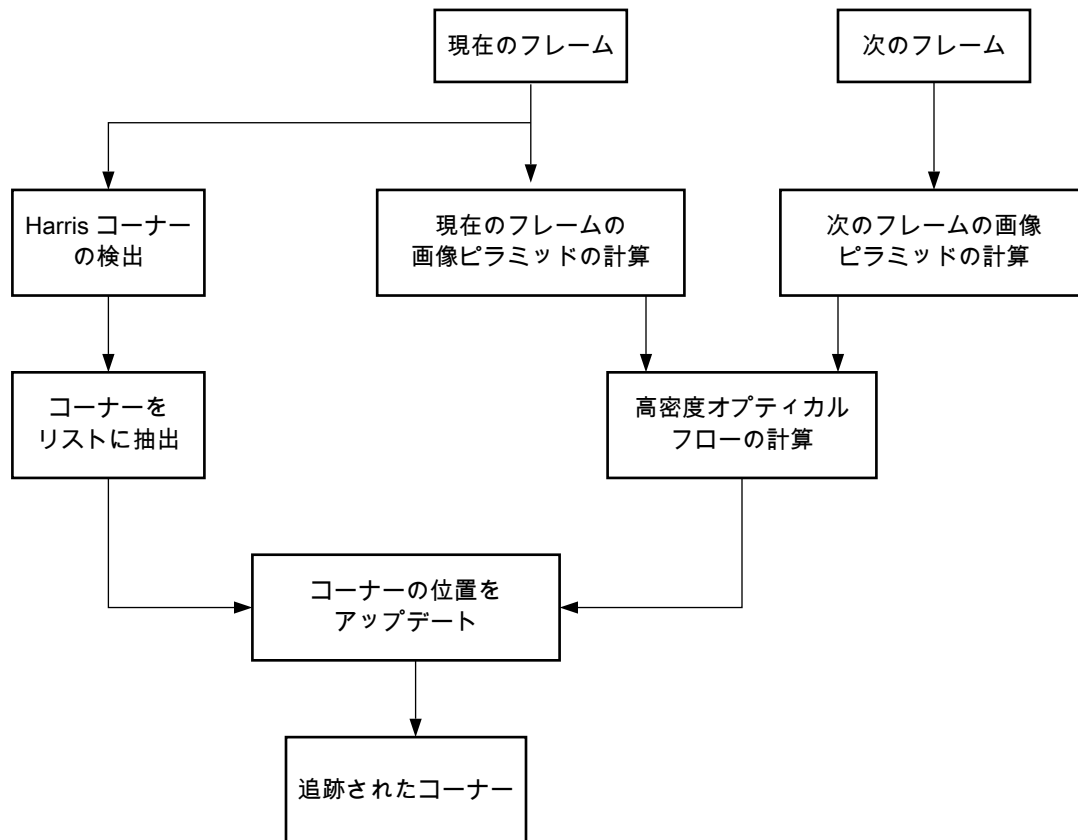
すべての入力データが準備でき、フラグがセットされると、ホスト プロセッサによりハードウェア関数が呼び出されます。ホスト関数では、最適化の問題を反復的に解決するため、ハードウェア関数へのフロー ベクター入力および出力がスワップされることに注意してください。また、`pyr_dense_optical_flow_accel()` 関数はハードウェア関数 `xf::densePyrOpticalFlow` の単なるラッパーです。ハードウェア関数へのテンプレート パラメーターは、このラッパー関数内で渡されます。

## 疎なオプティカル フローを使用したコーナー追跡

この例では、連続するビデオ フレームのセット内の特徴点を検出して追跡するところを示します。特徴検出には Harris コーナー検出器を使用し、追跡には Lucas-Kanade (LK) オプティカル フローを変更したものを使用しています。アルゴリズムの主要部分は、現在のフレームと次のフレームを入力として取り込み、追跡されたコーナーのリストを出力します。現在の画像がセット内の最初のフレームの場合、コーナー検出を実行し、追跡する特徴を検出します。追跡が必要な点を含むフレーム数も入力として供給します。

コーナー追跡の例では、xfOpenCV ライブラリからの 5 つのハードウェア関数 (`xf::cornerHarris`、`xf::cornersImgToList`、`xf::cornerUpdate`、`xf::pyrDown`、および `xf::densePyrOpticalFlow`) が使用されています。

図 12: 疎なオプティカル フローを使用したコーナー追跡



xf::densePyrOpticalFlow 関数の出力からの高密度フロー ベクターがまばらに選択されるようにするため、新しいハードウェア関数 xf::cornerUpdate が追加されています。これは、パイプラインの次の関数がメモリにランダムにアクセスして検索する必要がないようにするため実行されます。この関数は、Harris コーナー検出器からのコーナーと高密度ピラミッド型オプティカル フロー関数からの高密度オプティカル フローベクターを取り込み、高密度フローベクターを使用して入力コーナーを追跡して、アップデートされたコーナー位置を出力することにより、疎なオプティカル フローの動作を模倣しています。このハードウェア関数は、720p の画像に対して 300 MHz で実行されて 10,000 個のコーナーを処理します。パイプラインにレイテンシはほとんど追加されません。

## cornerUpdate()

### API 構文

```

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)

```

### パラメーターの説明

次の表に、テンプレートと関数のパラメーターを説明します。



表 445: cornerUpdate 関数のパラメーターの説明

パラメーター	説明
MAXCORNERSNO	関数で処理する必要がある最大コーナー数
TYPE	入力ピクセルのデータ型。8 ビット、符号なし、1 チャンネル (XF_8UC1) のみサポート。
ROWS	入力および出力画像の最大高さ (8 の倍数で指定)。
COLS	入力および出力画像の最大幅 (8 の倍数で指定)。
NPC	1 サイクルごとに処理されるピクセル数。XF_NPPC1 (1 サイクルごとに 1 ピクセル動作) のみサポート。
list_fix	16, 5 フォーマット (16 整数ビットと 5 小数ビット) で記述されたコーナー ロケーションのバックされた浮動小数点座標のリスト。ビット 20 ~ 0 は列番号、ビット 41 ~ 21 は行番号を示します。残りのビットはフラグとして使用されます。このフラグは、追跡されているコーナーが有効である場合にセットされます。
list	符号なしの short 型で記述されたコーナー ロケーションのバックされた正の short 整数座標のリスト。ビット 15 ~ 0 は列番号、ビット 31 ~ 16 は行番号を示します。このリストは、Harris コーナー検出器で出力されるリストと同じです。
nCorners	追跡するコーナーの数。
flow_vectors	xf::DensePyrOpticalFlow 関数のバックされたフロー ベクター
harris_flag	1 の場合、関数は入力コーナーを list から取得します。 0 の場合、関数は入力コーナーを list_fix から取得します。

例のコードは、xfOpenCV ライブラリを使用して読み出しおよび処理された入力ビデオで動作します。ホストでのコーナーの処理と追跡は、xf\_corner\_tracker\_accel() 関数で実行します。

## cornersImgToList()

### API 構文

```
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)
```

### パラメーターの説明

次の表に、テンプレートと Kintex® UltraScale+™ 関数のパラメーターを説明します。

表 446: CornerImgToList 関数のパラメーターの説明

パラメーター	説明
_src	Harris コーナー検出器の出力画像。この xf::Mat オブジェクトのサイズが Harris コーナー検出器への入力画像のサイズです。コーナーがその位置にある場合は各ピクセルの値は 255 となり、それ以外の場所は 0 になります。
list	Harris 検出器で検出されたコーナーを格納するために割り当てられた 32 ビット メモリ (MAXCORNERS のサイズ)
ncorners	Harris で検出されるコーナーの合計 (リスト内のコーナーの数)。

## cornerTracker()

xf\_corner\_tracker\_accel() 関数は、ホストでコーナーの処理と追跡を実行します。

## API 構文

```
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)
```

## パラメーターの説明

次の表に、テンプレートと関数のパラメーターを説明します。

パラメーター	説明
flow	ハードウェア関数を使用した反復計算中に、バックされたフロー ベクターを一時的に格納するために割り当てられた xf::Mat
flow_iter	ハードウェア関数を使用した反復計算中に、バックされたフロー ベクターを一時的に格納するために割り当てられた xf::Mat
mat_imagepyr1	xf::Mat のサイズがピラミッド レベル数に等しい配列。最初の画像のピラミッド レベル数を保存します。
mat_imagepyr2	xf::Mat のサイズがピラミッド レベル数に等しい配列。2 つ目の画像のピラミッド レベル数を保存します。
inHarris	Harris コーナー検出器への xf::Mat の入力画像
outHarris	Harris 検出器からの出力画像。コーナーがその位置にある場合は画像は 255 となり、それ以外の場所は 0 になります。
list	Harris 検出器で検出されたコーナーを格納するために割り当てられた 32 ビット メモリ (MAXCORNERS のサイズ)
listfixed	xf::cornerUpdate で追跡するコーナーを格納するために割り当てられた 64 ビット メモリ (MAXCORNERS のサイズ)
pyr_h	各ピラミッド レベルで画像の高さを格納するための、画像のピラミッド レベル数を含む整数の配列
pyr_w	各ピラミッド レベルで画像の幅を格納するための、画像のピラミッド レベル数を含む整数の配列
num_corners	Harris コーナー検出器で検出されたコーナーの数と同じサイズの配列
harrisThresh	Harris コーナー検出器 (xf::harris) へのしきい値入力
harris_flag	入力画像のセットに対して xf::harris で検出されたコーナーを使用するために、この関数の呼び出し元で使用されるフラグ

## 画像処理

次に、ハードウェアパイプラインでの画像処理の手順を示します。

1. xf::cornerharris を呼び出して最初の入力画像の処理を開始します。
2. xf::cornerHarris の出力が SDSoC™ によりハードウェアでパイプライン処理され、xf::cornersImgToList に送信されます。この関数は、コーナーが 255 か、それ以外の場合は 0 とマークされたコーナーを含む画像を取り込んで、それらをコーナーのリストに変換します。
3. 同時に、xf::pyrDown が 2 つの画像ピラミッドを作成し、反復ピラミッド型高密度オプティカル フローの例で説明されているように 2 つの画像ピラミッドを使用して高密度オプティカル フローを計算します。
4. xf::densePyrOpticalFlow を 2 つの画像ピラミッドと一緒に入力として呼び出します。

5. `xf::cornerUpdate` 関数を呼び出して、2 つ目の画像のコーナーを追跡します。`harris_flag` がイネーブルの場合 `cornerUpdate` はリストの出力からのコーナーを追跡し、それ以外の場合は前に追跡したコーナーを追跡します。

```
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;
    }
}
```

`xf_corner_tracker_accel()` 関数は、最初のフレームの処理中またはコーナーの再検出が必要なときにセットされる `harris_flag` というフラグを入力として使用します。`xf::cornerUpdate` 関数は、アップデートされたコーナーを `xf::cornerImgToList` の出力コーナー リストと同じメモリ ロケーションに出力します。つまり、`harris_flag` のセットが解除されたときの `xf::cornerUpdate` へのコーナー入力は、前のサイクルで追跡されたコーナー (現在の入力フレームの最初のフレームのコーナー) です。

高密度オプティカル フローが計算された後、`harris_flag` がセットされている場合は、`xf::cornerharris` で検出され、`xf::cornersImgToList` でアップデートされたコーナーの数が `num_corners` 変数にコピーされます。この変数は、`xf_corner_tracker_accel()` 関数の出力の 1 つです。もう 1 つの出力は、追跡されたコーナーのリスト `listfixed` です。`harris_flag` がセットされている場合は `xf::cornerUpdate` で `list` メモリ ロケーション内のコーナーが追跡され、セットされていない場合は `listfixed` メモリ ロケーション内のコーナーが追跡されます。

## 色検出

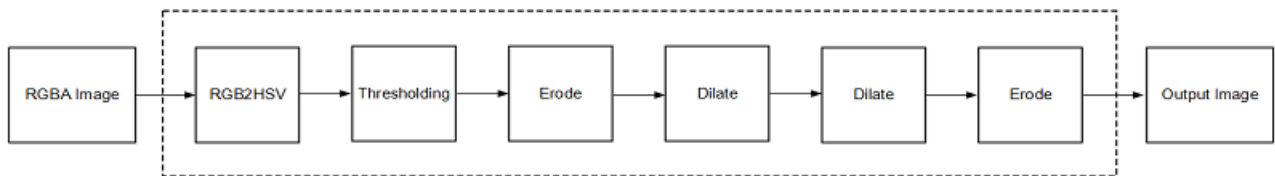
色検出アルゴリズムは、物体の色に基づいた色物体追跡および物体検出に使用されます。色に基づく方法は、物体と背景の色が大きく異なる場合に、物体の検出および分割に非常に有益です。

色検出の例では、xfOpenCV ライブラリから次の4つのハードウェア関数を使用します。

- xf::RGB2HSV
- xf::colorthresholding
- xf::erode
- xf::dilate

色検出の例では、まず元の BGR 画像の色空間を HSV 色空間に変換します。これは、HSV 色空間が色に基づく画像分割に最も適した色空間であるからです。その後、H (色相)、S (彩度)、および V (値) 値に基づいて HSV 画像に対してしきい値処理を実行し、255 または 0 を返します。画像のしきい値処理後、erode (モルフォロジカル オープニング) および dilate (モルフォロジカル クロージング) 関数を適用して画像の不要な白い斑点 (ノイズ) を削減します。この例では、erode および dilate 関数の 2 つのハードウェアインスタンスを使用して、erode の後に dilate を適用し、もう一度 dilate を適用してその後に erode を適用します。

図 13: 色検出



次の例に、色検出アルゴリズムを示します。

```

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);
}

```

この例では、ソース画像が `xf::RGB2HSV` 関数に渡され、その出力が `xf::colorthresholding` モジュールに渡され、しきい値処理された画像が `xf::erode` 関数、そして `xf::dilate` 関数に渡されて、最終的な出力画像が返されます。

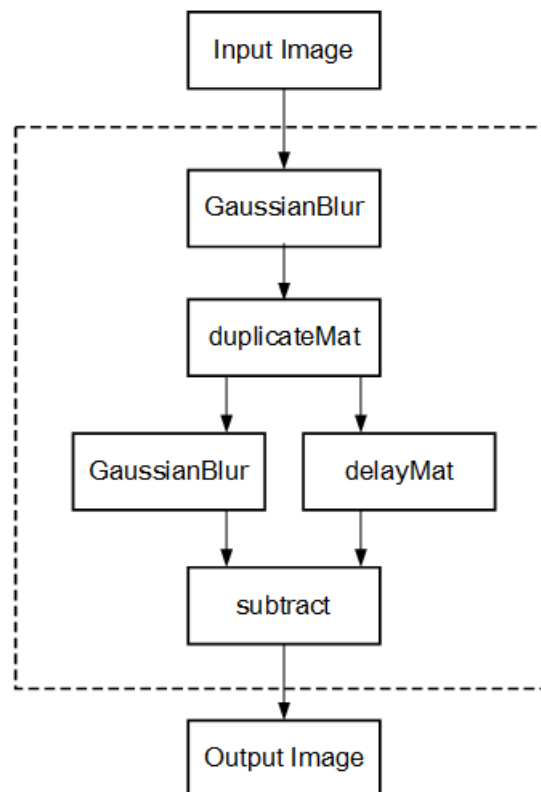
## ガウシアン フィルターの差

ガウシアン フィルターの差の例では、xfOpenCV ライブラリから次の 4 つのハードウェア関数を使用します。

- `xf::GaussianBlur`
- `xf::duplicateMat`
- `xf::delayMat`
- `xf::subtract`

ガウシアン フィルターの差関数は、元のソース画像にガウシアン フィルターを適用し、そのガウシアンぼけ画像を複製して 2 つの画像とします。複製された画像の 1 つに `GaussianBlur` 関数を適用し、もう 1 つの画像はそのまま保存します。その後、ガウシアンが 2 回適用された画像と複製されたもう 1 つの画像に対して減算関数を実行します。この時点で、複製されたもう 1 つの画像は、もう 1 つの画像に 2 回目のガウシアンが適用されて少なくとも 1 ピクセルの出力が生成されるまで待機する必要があります。そのため、`xf::delayMat` 関数を使用して遅延を追加します。

図 14: ガウシアン フィルターの差



次に、ガウシアン フィルターの差の例を示します。

```
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);
}
```

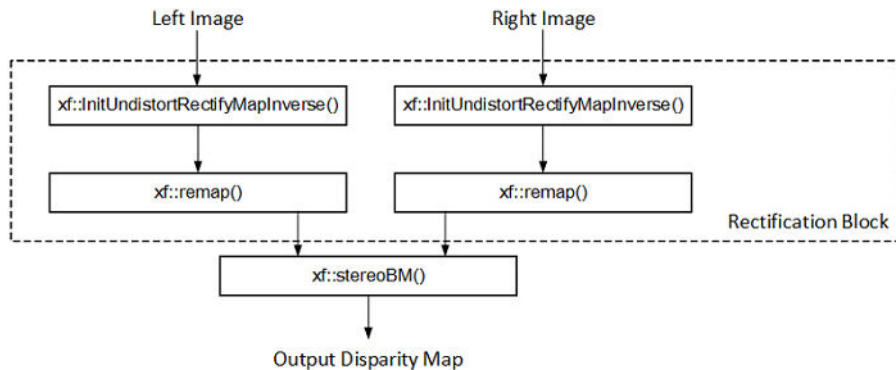
この例では、ソース画像 `imginput` に `GaussianBlur` 関数が適用され、その結果の画像 `imgin1` が `xf::duplicateMat` に渡されます。`imgin2` と `imgin3` は、ガウシアンが適用された画像の複製です。`imgin2` にはもう一度 `GaussianBlur` が適用され、その結果が `imgin4` に保存されます。その後、`imgin4` と `imgin3` の間で減算が実行されます。ただし、`imgin3` は `imgin4` の少なくとも 1 ピクセルが生成されるまで待機する必要があります。そのため `imgin3` に遅延が適用され、`imgin5` に保存されます。最後に `imgin4` と `imgin5` の間で減算が実行されます。

## ステレオ ビジョン パイプライン

視差マップの生成は、環境の 3 次元マップを作成する最初の手順の 1 つです。xfOpenCV ライブラリには、ステレオカメラからのカメラ パラメーターおよび入力を取り込んで視差マップを計算する画像処理パイプラインをビルドするためのコンポーネントが含まれています。

パイプラインに関連する 2 つの主なコンポーネントは、ステレオ平行化とローカル ブロック マッチング方法を使用した視差推定です。ローカル ブロック マッチングを使用した視差推定は xfOpenCV の個別のコンポーネントですが、平行化ブロックは `xf::InitUndistortRectifyMapInverse()` と `xf::Remap()` を使用して作成できます。次に、データフローパイプラインを示します。カメラのパラメーターは、パイプラインへの追加の入力です。

図 15: ステレオ ビジョン パイプライン



次に、パイプラインのコードを示します。

```

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, mapxLMat, mapyLMat, _cm_size, _dc_size);

    xf::remap<XF_REMAP_BUFSIZE, XF_INTERPOLATION_BILINEAR, XF_8UC1, XF_32FC1, XF_8UC1, 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, mapxRMat, mapyRMat, _cm_size, _dc_size);

    xf::remap<XF_REMAP_BUFSIZE, XF_INTERPOLATION_BILINEAR, XF_8UC1, XF_32FC1, XF_8UC1, XF_HEIGHT, XF_WIDTH, XF_NPPC1, XF_USE_URAM>(rightMat, rightRemappedMat, mapxRMat, mapyRMat);

    xf::StereoBM<SAD_WINDOW_SIZE, NO_OF_DISPARITIES, PARALLEL_UNITS, XF_8UC1, XF_16UC1, XF_HEIGHT, XF_WIDTH, XF_NPPC1, XF_USE_URAM>(leftRemappedMat, rightRemappedMat, dispMat, bm_state);
}

```

# その他のリソースおよび法的通知

## ザイリンクス リソース

アンサー、資料、ダウンロード、フォーラムなどのサポートリソースは、[ザイリンクス サポート](#) サイトを参照してください。

## Documentation Navigator およびデザイン ハブ

ザイリンクス Documentation Navigator (DocNav) では、ザイリンクスの資料、ビデオ、サポート リソースにアクセスでき、特定の情報を取得するためにフィルター機能や検索機能を利用できます。DocNav を開くには、次のいずれかを実行します。

- Vivado<sup>®</sup> IDE で [Help] → [Documentation and Tutorials] をクリックします。
- Windows で [スタート] → [すべてのプログラム] → [Xilinx Design Tools] → [DocNav] をクリックします。
- Linux コマンド プロンプトに「docnav」と入力します。

ザイリンクス デザイン ハブには、資料やビデオへのリンクがデザイン タスクおよびトピックごとにまとめられており、これらを参照することでキー コンセプトを学び、よくある質問 (FAQ) を参考に問題を解決できます。デザイン ハブにアクセスするには、次のいずれかを実行します。

- DocNav で [Design Hub View] タブをクリックします。
- ザイリンクス ウェブサイトで[デザイン ハブ](#) ページを参照します。

**注記:** DocNav の詳細は、ザイリンクス ウェブサイトの [Documentation Navigator](#) ページを参照してください。



**注意:** DocNav からは、日本語版は参照できません。ウェブサイトのデザイン ハブ ページをご利用ください。

## 参考資料

このガイドの補足情報は、次の資料を参照してください。

日本語版のバージョンは、英語版より古い場合があります。

1. 『SDSoC 環境リリース ノート、インストール、およびライセンスガイド』 ([UG1294](#))



2. 『SDSoC 環境ユーザー ガイド』 ([UG1027](#))
3. 『SDSoC 環境チュートリアル: 概要』 ([UG1028](#))
4. 『SDSoC 環境チュートリアル: プラットフォームの作成』 ([UG1236](#))
5. 『SDSoC 環境プラットフォーム開発ガイド』 ([UG1146](#))
6. 『SDSoC 環境プロファイリングおよび最適化ガイド』 ([UG1235](#))
7. 『SDx コマンドおよびユーティリティ リファレンス ガイド』 ([UG1279](#))
8. 『SDSoC 環境プログラマ ガイド』 ([UG1278](#))
9. 『SDSoC 環境デバッグ ガイド』 ([UG1282](#))
10. 『SDx プラグマ リファレンス ガイド』 ([UG1253](#))
11. 『UltraFast エンベデッド デザイン設計手法ガイド』 ([UG1046](#): [英語版](#)、[日本語版](#))
12. 『Zynq-7000 SoC ソフトウェア開発者向けガイド』 ([UG821](#): [英語版](#)、[日本語版](#))
13. 『Zynq UltraScale+ MPSoC: ソフトウェア開発者向けガイド』 ([UG1137](#): [英語版](#)、[日本語版](#))
14. 『Zynq-7000 XC7Z020 SoC 用 ZC702 評価ボード ユーザー ガイド』 ([UG850](#))
15. 『ZCU102 評価ボード ユーザー ガイド』 ([UG1182](#))
16. 『PetaLinux ツール資料: リファレンス ガイド』 ([UG1144](#))
17. 『Vivado Design Suite ユーザー ガイド: 高位合成』 ([UG902](#))
18. 『Vivado Design Suite ユーザー ガイド: カスタム IP の作成とパッケージ』 ([UG1118](#))
19. [SDSoC 開発環境ウェブ ページ](#)
20. [Vivado® Design Suite 資料](#)

## お読みください: 重要な法的通知

本通知に基づいて貴殿または貴社(本通知の被通知者が個人の場合には「貴殿」、法人その他の団体の場合には「貴社」。以下同じ)に開示される情報(以下「本情報」といいます)は、サイリンクスの製品を選択および使用することのためにのみ提供されます。適用される法律が許容する最大限の範囲で、(1)本情報は「現状有姿」、およびすべて受領者の責任で(with all faults)という状態で提供され、サイリンクスは、本通知をもって、明示、黙示、法定を問わず(商品性、非侵害、特定目的適合性の保証を含みますがこれらに限られません)、すべての保証および条件を負わない(否認する)ものとします。また、(2)サイリンクスは、本情報(貴殿または貴社による本情報の使用を含む)に関係し、起因し、関連する、いかなる種類・性質の損失または損害についても、責任を負わない(契約上、不法行為上(過失の場合を含む)、その他のいかなる責任の法理によるかを問わない)ものとし、当該損失または損害には、直接、間接、特別、付随的、結果的な損失または損害(第三者が起こした行為の結果被った、データ、利益、業務上の信用の損失、その他あらゆる種類の損失や損害を含みます)が含まれるものとし、それは、たとえ当該損害や損失が合理的に予見可能であったり、サイリンクスがそれらの可能性について助言を受けていた場合であったとしても同様です。サイリンクスは、本情報に含まれるいかなる誤りも訂正する義務を負わず、本情報または製品仕様のアップデートを貴殿または貴社に知らせる義務も負いません。事前の書面による同意のない限り、貴殿または貴社は本情報を再生産、変更、頒布、または公に展示してはなりません。一定の製品は、サイリンクスの限定的保証の諸条件に従うこととなるので、<https://japan.xilinx.com/legal.htm#tos> で見られるサイリンクスの販売条件を参照してください。IP コアは、サイリンクスが貴殿または貴社に付与したライセンスに含まれる保証と補助的条件に従うことになります。サイリンクスの製品は、フェイルセーフとして、または、フェイルセーフの動作を要求するアプリケーションに使用するために、設計されたり意図されたりしていません。そのような重大なアプリケーションにサイリンクスの製品を使用する場合のリスクと責任は、貴殿または貴社が単独で負うものです。<https://japan.xilinx.com/legal.htm#tos> で見られるサイリンクスの販売条件を参照してください。

### 自動車用のアプリケーションの免責条項

オートモーティブ製品 (製品番号に「XA」が含まれる) は、ISO 26262 自動車用機能安全規格に従った安全コンセプトまたは余剰性の機能 (「セーフティ設計」) がない限り、エアバッグの展開における使用または車両の制御に影響するアプリケーション (「セーフティ アプリケーション」) における使用は保証されていません。顧客は、製品を組み込むすべてのシステムについて、その使用前または提供前に安全を目的として十分なテストを行うものとし、セーフティ設計なしにセーフティ アプリケーションで製品を使用するリスクはすべて顧客が負い、製品責任の制限を規定する適用法令および規則にのみ従うものとし、

### 商標

© Copyright 2017-2019 Xilinx, Inc. Xilinx、Xilinx のロゴ、Alveo、Artix、Kintex、Spartan、Versal、Virtex、Vivado、Zynq、およびこの文書に含まれるその他の指定されたブランドは、米国およびその他の各国のザイリックス社の商標です。OpenCL および OpenCL のロゴは Apple Inc. の商標であり、Khronos による許可を受けて使用されています。HDMI、HDMI のロゴ、および High-Definition Multimedia Interface は、HDMI Licensing LLC の商標です。AMBA、AMBA Designer、Arm、ARM1176JZ-S、CoreSight、Cortex、PrimeCell、Mali、および MPCore は、EU およびその他の各国の Arm Limited の商標です。すべてのその他の商標は、それぞれの所有者に帰属します。

この資料に関するフィードバックおよびリンクなどの問題につきましては、[jpn\\_trans\\_feedback@xilinx.com](mailto:jpn_trans_feedback@xilinx.com) まで、または各ページの右下にある [フィードバック送信] ボタンをクリックすると表示されるフォームからお知らせください。フィードバックは日本語で入力可能です。いただきましたご意見を参考に早急に対応させていただきます。なお、このメール アドレスへのお問い合わせは受け付けておりません。あらかじめご了承ください。