XILINX
ALL PROGRAMMABLE™

Video Processing Design Using HLS

Alex Paek
Sept 2015

Accelerating Smart Vision Applications

HD Surveillance

Driver Assistance

OpenCV

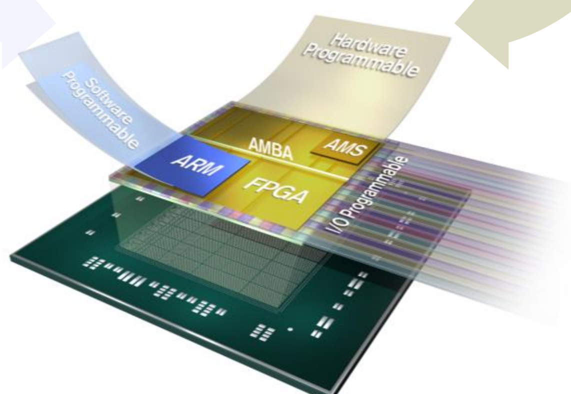
Frame-level processing
Library for CPU

Pixel processing and basic
functions for analytics

VIVADO HLS

Video Conferencing

Machine Vision



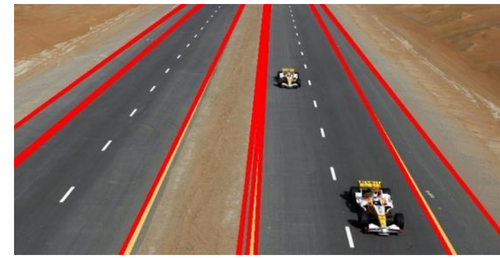
ZYNQ

A&D UAV

Office-class MFP

Image Processing Example: Driver Assistance

- Analyze a video frame to detect road lane markings

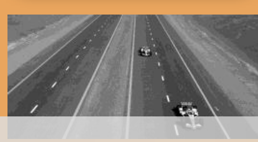


Lane Detection → Lane Tracking → Decision Making

Processing Rate for Real-Time Execution

Requires Hardware | Can be Performed in Software

Conversion → Histogram Equalization → Detection → Transform



Lane Detection

Optimal Solution is a Mix of Hardware and Software

HLS Video Library

- **C video libraries**
  - Available within Vivado HLS tool header files
    - *hls_video.h* library
    - *hls_opencv.h* library – test bench purpose only
- **Enable migration of OpenCV designs for use with the Vivado HLS tool**

HLS Video Library

- **C++ code contained in `hls` namespace:**
  `#include "hls_video.h"`

- **Similar interface; equivalent behavior with OpenCV**
  – OpenCV library: `cvScale(src, dst, scale, shift);`
  – HLS video library: `hls::Scale<...>(src, dst, scale, shift);`

- **Some constructor arguments have corresponding or replacement template parameters**
  – OpenCV library: `cv::Mat mat(rows, cols, CV_8UC3);`
  – HLS video library: `hls::Mat<ROWS,COLS,HLS_8UC3> mat(rows,cols);`
    - ROWS and COLS specify the maximum size of an image processed

HLS Video Functions

Video Data Modeling	
Linebuffer class	Window class

AXI4-Stream IO Functions	
AXIvideo2Mat	Mat2AXIvideo

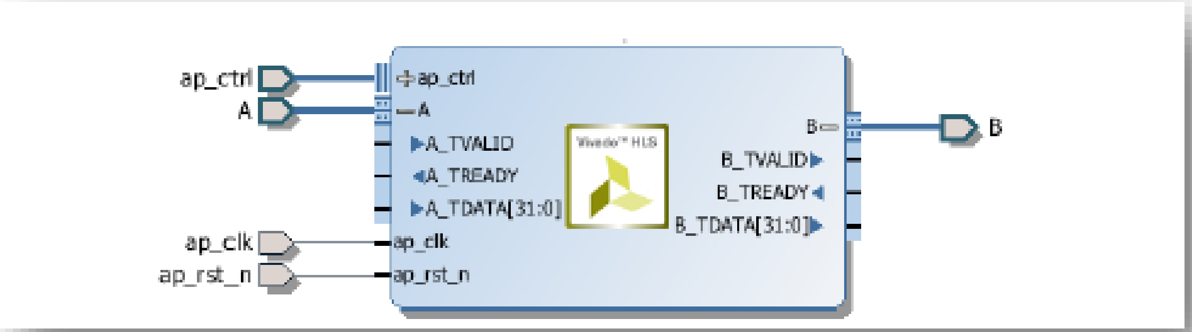
OpenCV Interface Functions			
cvMat2AXIvideo	AXIvideo2cvMat	cvMat2hlsMat	hlsMat2cvMat
IplImage2AXIvideo	AXIvideo2IplImage	IplImage2hlsMat	hlsMat2IplImage
CvMat2AXIvideo	AXIvideo2CvMat	CvMat2hlsMat	hlsMat2CvMat

Video Functions			
AbsDiff	Duplicate	MaxS	Remap
AddS	EqualizeHist	Mean	Resize
AddWeighted	Erode	Merge	Scale
And	FASTX	Min	Set
Avg	Filter2D	MinMaxLoc	Sobel
AvgSdv	GaussianBlur	MinS	Split
Cmp	Harris	Mul	SubRS
CmpS	HoughLines2	Not	SubS
CornerHarris	Integral	PaintMask	Sum
CvtColor	InitUndistortRectifyMap	Range	Threshold
Dilate	Max	Reduce	Zero

For function signatures and descriptions, see the HLS user guide

AXI Interface for Video Design

- **AXI Streaming**
  - Typically used for streaming video
  - requires fifo based interface
  - can be applied to array and pointer data types
  - side channel support to mark lines and frames



- **AXI Lite Slave**
  - Typically used for control and setup

AXI Streaming for Video

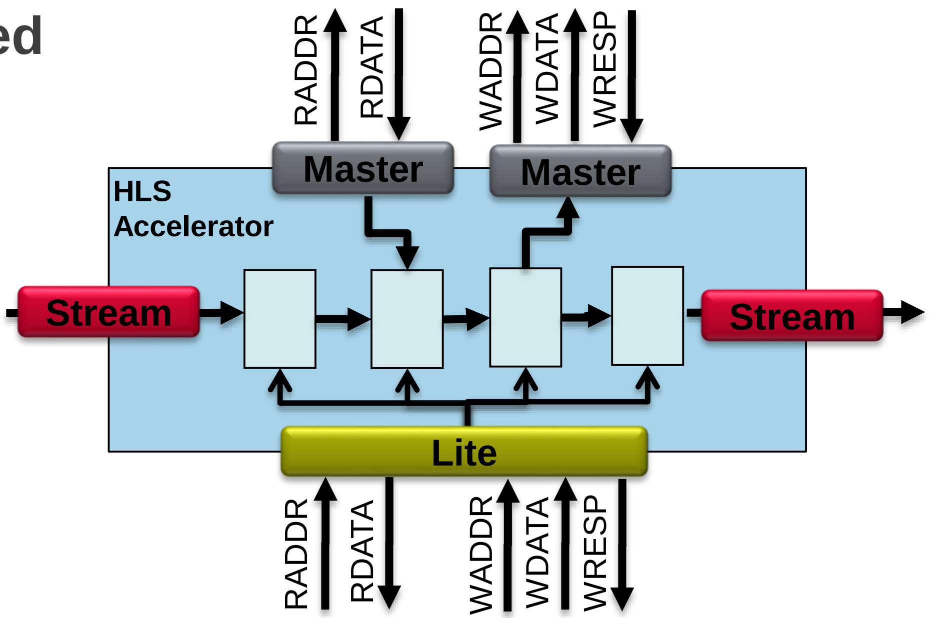
- **AXI Streaming Slave Video Protocol**

Function	Width	Direction	AXI4-Stream Signal Name	Video Specific Name
Video Data	8, 16, 24, 32, 40, 48, 56, 64	IN	s_axis_video_tdata	DATA
Valid	1	IN	s_axis_video_tvalid	VALID
Ready	1	OUT	s_axis_video_tready	READY
Start Of Frame	1	IN	s_axis_video_tuser	SOF
End Of Line	1	IN	s_axis_video_tlast	EOL

- **AXI Streaming Master Video Protocol**
  – The same except the direction of input and output

Using AXI Interface

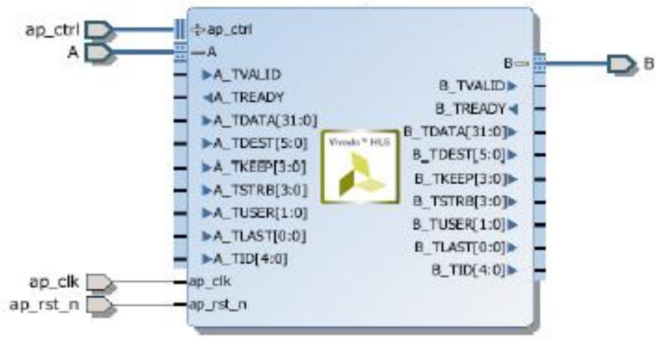
- **All 3 AXI interfaces are supported by the INTERFACE directive**
  - AXI4-Master
  - AXI4-Lite (Slave)
  - AXI4-Stream
- **Provided in RTL after Synthesis (not Export)**
- **Supported by the C/RTL co-simulation**



Directives for AXI Streaming

- **Use the HLS INTERFACE axis directive to specify an AXI STREAM**

```
void example(int A[50], int B[50]) {
    //Set the HLS native interface types
    #pragma HLS INTERFACE axis port=A
    #pragma HLS INTERFACE axis port=B
```



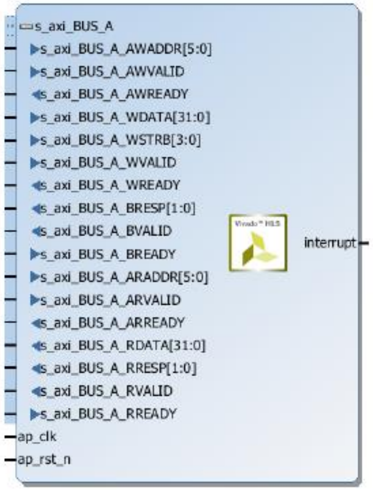
- **Group multiple variables to same stream by assigning to same bus_bundle name**

```
#pragma HLS INTERFACE axis port=A bus_bundle A
#pragma HLS INTERFACE axis port=B bus_bundle B
```

Directives for AXI Lite

- **Use the HLS INTERFACE s_axilite directive to specify an AXI Lite interface**

```
void example(char *a, char *b, char *c)
{
        #pragma HLS INTERFACE s_axilite port=return bundle=BUS_A
        #pragma HLS INTERFACE s_axilite port=a bundle=BUS_A
        #pragma HLS INTERFACE s_axilite port=b bundle=BUS_A
        #pragma HLS INTERFACE s_axilite port=c bundle=BUS_A
*c += *a + *b;
}
```



AXI Lite Drivers

- **C driver files for AXI Lite interface**

  - created in solution/imp/drivers dir

File Path	Usage Mode	Description
data/example.mdd	Standalone	Driver definition file. When exporting a Pcore, this file is named example_top_v2_1_0.mdd.
data/example.tcl	Standalone	Used by SDK to integrate the software into an SDK project. When exporting a Pcore, this file is named example_top_v2_1_0.tcl.
src/xexample_hw.h	Both	Defines address offsets for all internal registers.
src/xexample.h	Both	API definitions
src/xexample.c	Both	Standard API implementations
src/xexample_sinit.c	Standalone	Initialization API implementations
src/xexample_linux.c	Linux	Initialization API implementations
src/Makefile	Standalone	Makefile

AXI Lite Drivers

- **xexample_hw.h contains complete list of memory mapped locations**

```
// 0x00 : Control signals
//        bit 0  - ap_start (Read/Write/SC)
//        bit 1  - ap_done (Read/COR)
//        bit 2  - ap_idle (Read)
//        bit 3  - ap_ready (Read)
//        bit 7  - auto_restart (Read/Write)
//        others - reserved
// 0x04 : Global Interrupt Enable Register
//        bit 0  - Global Interrupt Enable (Read/Write)
//        others - reserved
// 0x08 : IP Interrupt Enable Register (Read/Write)
//        bit 0  - Channel 0 (ap_done)
//        others - reserved
// 0x0c : IP Interrupt Status Register (Read/TOW)
//        bit 0  - Channel 0 (ap_done)
//        others - reserved
// 0x10 : Data signal of a
//        bit 7~0 - a[7:0] (Read/Write)
//        others  - reserved
```

C Testbench

- **Interface libraries convert to/from OpenCV image to HLS type**
  - HLS MAT format: synthesizable and AXI4 Stream support

```
#include "hls_opencv.h"
//Top Level C Function
int main (int argc, char** argv) {

    IplImage* src = cvLoadImage(INPUT_IMAGE);
    IplImage* dst = cvCreateImage(cvGetSize(src), src->depth, src->nChannels);

    AXI_STREAM  src_axi, dst_axi;
    IplImage2AXIvideo(src, src_axi);

    image_filter(src_axi, dst_axi, src->height, src->width);

    AXIvideo2IplImage(dst_axi, dst);

    cvSaveImage(OUTPUT_IMAGE, dst);
```

HLS Video Libraries

Standard OpenCV files, formats & types

Convert to Xilinx AXI4 Video Stream

Function to Synthesize

Convert Xilinx AXI4 Video Stream back to OpenCV types

C Function to Synthesize (DUT)

```
#include "hls_video.h"
#include "ap_axi_sdata.h";
//Top Level C Function for Synthesis
void image_filter(AXI_STREAM& inter_pix, AXI_STREAM& out_pix, int rows, int cols) {
    //Create AXI streaming interfaces for the core

    RGB_IMAGE img_0(rows, cols);
    ..etc..
    RGB_IMAGE img_5(rows, cols);
    RGB_PIXEL pix(50, 50, 50);
#pragma HLS dataflow
    hls::AXIvideo2Mat(inter_pix, img_0);

    hls::Sobel(img_0, img_1, 1, 0);
    hls::SubS(img_1, pix, img_2);
    hls::Scale(img_2, img_3, 2, 0);
    hls::Erode(img_3, img_4);
    hls::Dilate(img_4, img_5);

    hls::Mat2AXIvideo(img_5, out_pix);

}
```

HLS Video & AXI Struct Libraries

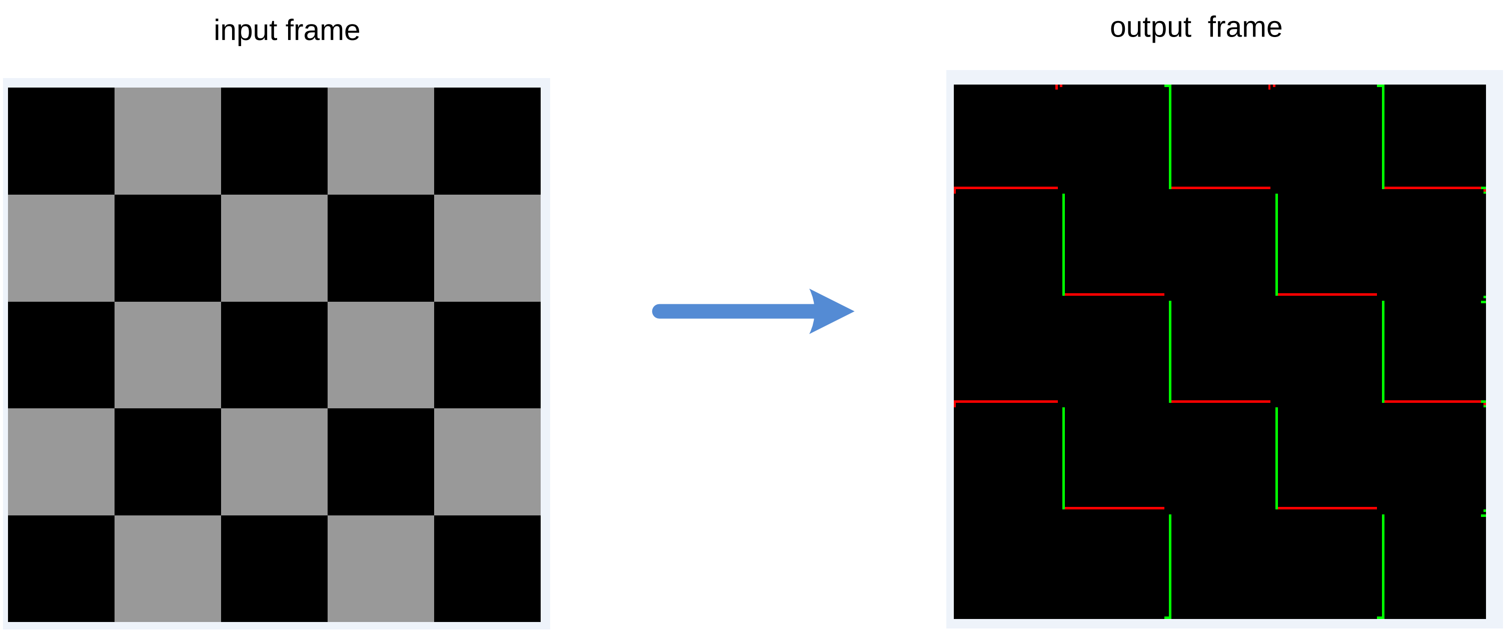
Convert Xilinx AXI4 Video Stream to HLS Mat data type

HLS Video functions are drop-in replacement for OpenCV function & provide high QoR

Convert HLS Mat type to Xilinx AXI4 Video Stream

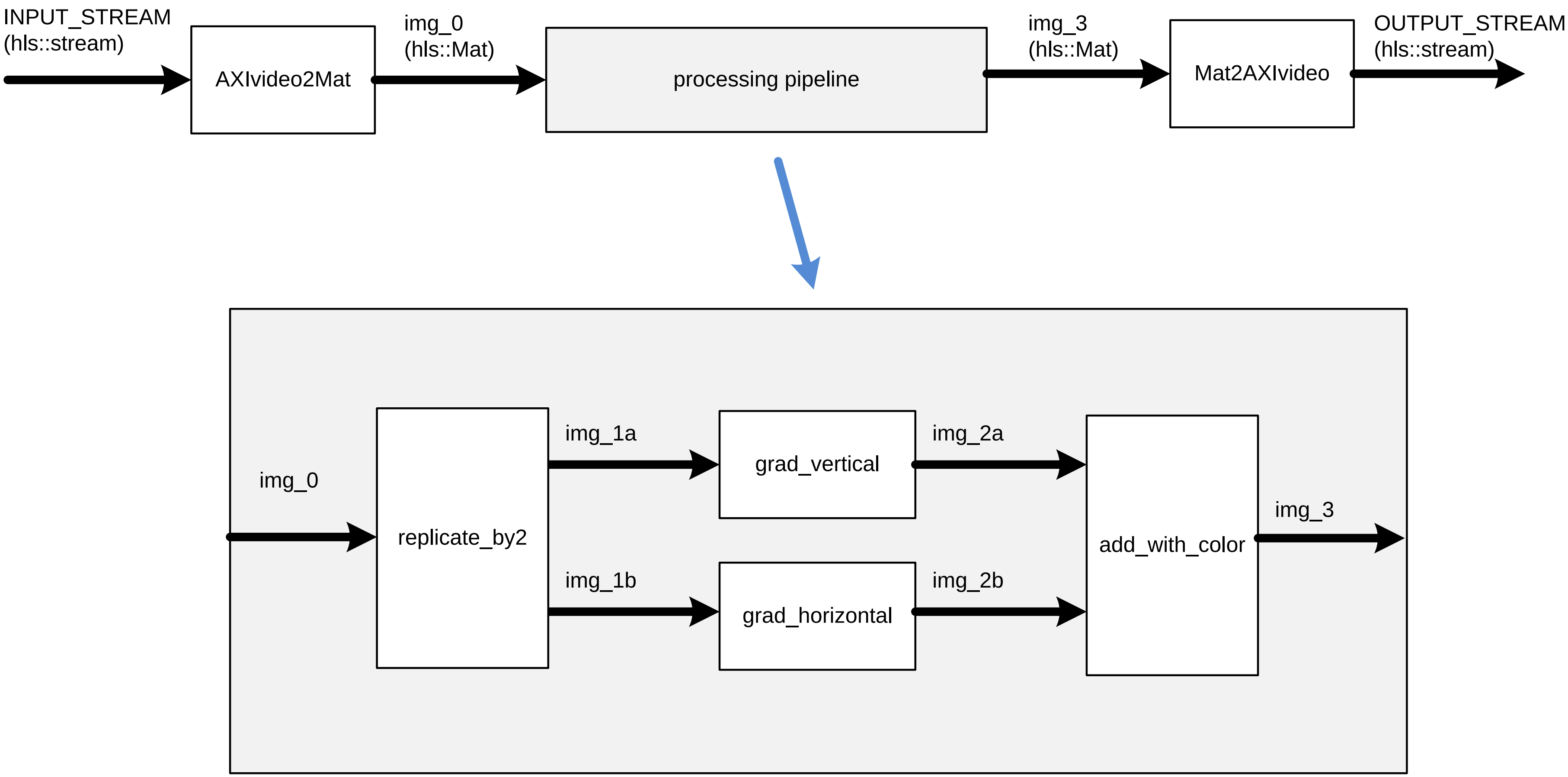
Video Processing Design

- 1080p frame at >60FPS. Pixel coming in at a rasterized streaming fashion
- Detect edges – color green for vertical edge, red for horizontal edges
- Target device: Zynq 7020 -2
- Pixel rate = Clock rate < 6 ns (= 167 MHz)



Block Diagram

- .



DUT code

- The top level code (image_filter.cpp) and head file (image_filter.h)
- “hls::Mat” structure can be viewed as a virtual frame buffer.

```
#include "image_filter.h"

//________________________________________________________________________________

void image_filter(AXI_STREAM& INPUT_STREAM, AXI_STREAM& OUTPUT_STREAM) {

//#pragma HLS INTERFACE axis depth=10000 port=INPUT_STREAM bundle=VIDEO_IN
//#pragma HLS INTERFACE axis depth=10000 port=OUTPUT_STREAM bundle=VIDEO_OUT
#pragma HLS INTERFACE axis port=INPUT_STREAM bundle=VIDEO_IN
#pragma HLS INTERFACE axis port=OUTPUT_STREAM bundle=VIDEO_OUT
//#pragma HLS INTERFACE s_axilite port=return   bundle=CONTROL_BUS

//#pragma HLS INTERFACE s_axilite port=rows     bundle=CONTROL_BUS //offset=0x14
//#pragma HLS INTERFACE s_axilite port=cols     bundle=CONTROL_BUS //offset=0x1C

#pragma HLS dataflow

//assert(rows <= MAX_HEIGHT);
//assert(cols <= MAX_WIDTH);
const int rows = MAX_HEIGHT;
const int cols = MAX_WIDTH;

RGB_IMAGE  img_0  (rows, cols);
RGB_IMAGE  img_1a (rows, cols);
RGB_IMAGE  img_1b (rows, cols);
RGB_IMAGE  img_2a (rows, cols);
RGB_IMAGE  img_2b (rows, cols);
RGB_IMAGE  img_3  (rows, cols);

// Convert AXI4 Stream data to hls::mat format
hls::AXIvideo2Mat(INPUT_STREAM, img_0);

// copy to 2 channels
replicate_by2<RGB_IMAGE,RGB_PIXEL>(img_0, img_1a, img_1b, rows, cols);

// gradient
grad_vertical<RGB_IMAGE>(img_1a, img_2a, rows, cols);
grad_horizontal<RGB_IMAGE>(img_1b, img_2b, rows, cols);

// combine
// combine2<RGB_IMAGE,RGB_PIXEL>(img_2a, img_2b, img_3, rows, cols);
add_with_color<RGB_IMAGE,RGB_PIXEL>(img_2a, img_2b, img_3, rows, cols);

// Convert the hls::mat format to AXI4 Stream format with SOF, EOL signals
hls::Mat2AXIvideo(img_3, OUTPUT_STREAM);

}
```

```
// maximum image size
#define MAX_WIDTH  1920
#define MAX_HEIGHT 1080

#include  "hls_video.h"
#include <ap_fixed.h>

typedef hls::stream<ap_axiu<32,1,1,1> >          AXI_STREAM;
//typedef hls::stream<ap_axiu<8,1,1,1> >           AXI_STREAM;

// check include/hls/hls_video_types.h for detail on pixel type
typedef hls::Mat<MAX_HEIGHT,   MAX_WIDTH,   HLS_8UC3> RGB_IMAGE;
typedef hls::Mat<MAX_HEIGHT/2, MAX_WIDTH/2, HLS_8UC3> RGB_IMAGE_HALF;
typedef hls::Scalar<3, unsigned char>                 RGB_PIXEL;
```

Using filter2D

```
//________________________________________________________________
// gradient vertical
//
template<typename IMG_T>
void grad_vertical(IMG_T& img_in, IMG_T& img_out, int rows, int cols) {

// 2D kernel for veritical gradient
const COEF_T coef_v[KS][KS]= {
  {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}
};
hls::Window<KS,KS,COEF_T> Sv;
for (int r=0; r<KS; r++) for (int c=0; c<KS; c++) Sv.val[r][c] = coef_v[r][c];

// point
hls::Point_<INDEX_T> anchor;
anchor.x=-1;
anchor.y=-1;

hls::Filter2D <hls::BORDER_CONSTANT> (img_in, img_out, Sv, anchor);

}
```

hls::Filter2D

Synopsis

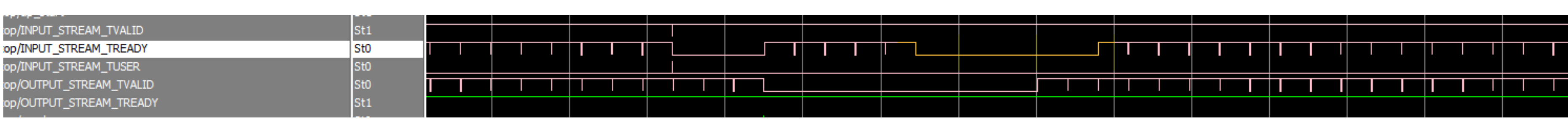
```
template<typename BORDERMODE, int SRC_T, int DST_T, typename KN_T, typename POINT_T,
int IMG_HEIGHT,int IMG_WIDTH,int K_HEIGHT,int K_WIDTH>
void Filter2D(
        Mat<IMG_HEIGHT, IMG_WIDTH, SRC_T>&_src,
        Mat<IMG_HEIGHT, IMG_WIDTH, DST_T>  &_dst,
        Window<K_HEIGHT,K_WIDTH,KN_T>&_kernel,
        Point_<POINT_T>anchor)

template<int SRC_T, int DST_T, typename KN_T, typename POINT_T,
int IMG_HEIGHT,int IMG_WIDTH,int K_HEIGHT,int K_WIDTH>
void Filter2D(
        Mat<IMG_HEIGHT, IMG_WIDTH, SRC_T>&_src,
        Mat<IMG_HEIGHT, IMG_WIDTH, DST_T>  &_dst,
        Window<K_HEIGHT,K_WIDTH,KN_T>&_kernel,
        Point_<POINT_T>anchor);
```

Parameters

Table 4-32: **Parameters**

Parameter	Description
src	Input image
dst	Output image
kernel	Kernel of 2D filtering, defined by hls::Window class
anchor	Anchor of the kernel that indicates that the relative position of a filtered point within the kernel

- **One issue: INPUT_STREAM_TREADY goes low during the beginning of each frame – active region, not the blanking region; thus, need about 6 lines of fifo to buffer the incoming pixels. CR Filed.**
- **Workaround: use border_mode = BORDER_CONSTANT**



Synthesis Results

Performance Estimates

Timing (ns)

Summary

Clock	Target	Estimated	Uncertainty
ap_clk	6.00	5.25	0.75

Latency (clock cycles)

Summary

Latency		Interval		
min	max	min	max	Type
4148543	4148543	2105198	2105198	dataflow

Detail

Instance

		Latency		Interval		
Instance	Module	min	max	min	max	Type
grp_image_filter_AXIvideo2Mat_fu_208	image_filter_AXIvideo2Mat	2077923	2077923	2077923	2077923	none
grp_image_filter_replicate_by2_Mat_Scalar_s_fu_263	image_filter_replicate_by2_Mat_Scalar_s	2073599	2073600	2073599	2073600	none
grp_image_filter_grad_vertical_Mat_s_fu_184	image_filter_grad_vertical_Mat_s	2105197	2105197	2105197	2105197	none
grp_image_filter_grad_horizontal_Mat_s_fu_196	image_filter_grad_horizontal_Mat_s	2105197	2105197	2105197	2105197	none
grp_image_filter_add_with_color_Mat_Scalar_s_fu_229	image_filter_add_with_color_Mat_Scalar_s	2073602	2073603	2073600	2073600	loop rewind
grp_image_filter_Mat2AXIvideo_fu_242	image_filter_Mat2AXIvideo	2076841	2076841	2076841	2076841	none

Synthesis Results

Utilization Estimates

Summary

Name	BRAM_18K	DSP48E	FF	LUT
DSP	-	-	-	-
Expression	-	-	-	-
FIFO	0	-	90	360
Instance	24	150	8655	4186
Memory	-	-	-	-
Multiplexer	-	-	-	-
Register	-	-	12	-
Total	24	150	8757	4546
Available	280	220	106400	53200
Utilization (%)	8	68	8	8

Interface

Summary

RTL Ports	Dir	Bits	Protocol	Source Object	C Type
INPUT_STREAM_TDATA	in	32	axis	VIDEO_IN_V_data_V	pointer
INPUT_STREAM_TKEEP	in	4	axis	VIDEO_IN_V_keep_V	pointer
INPUT_STREAM_TSTRB	in	4	axis	VIDEO_IN_V_strb_V	pointer
INPUT_STREAM_TUSER	in	1	axis	VIDEO_IN_V_user_V	pointer
INPUT_STREAM_TLAST	in	1	axis	VIDEO_IN_V_last_V	pointer
INPUT_STREAM_TID	in	1	axis	VIDEO_IN_V_id_V	pointer
INPUT_STREAM_TDEST	in	1	axis	VIDEO_IN_V_dest_V	pointer
INPUT_STREAM_TVALID	in	1	axis	VIDEO_IN_V_dest_V	pointer
INPUT_STREAM_TREADY	out	1	axis	VIDEO_IN_V_dest_V	pointer
OUTPUT_STREAM_TDATA	out	32	axis	VIDEO_OUT_V_data_V	pointer
OUTPUT_STREAM_TKEEP	out	4	axis	VIDEO_OUT_V_keep_V	pointer
OUTPUT_STREAM_TSTRB	out	4	axis	VIDEO_OUT_V_strb_V	pointer
OUTPUT_STREAM_TUSER	out	1	axis	VIDEO_OUT_V_user_V	pointer
OUTPUT_STREAM_TLAST	out	1	axis	VIDEO_OUT_V_last_V	pointer
OUTPUT_STREAM_TID	out	1	axis	VIDEO_OUT_V_id_V	pointer
OUTPUT_STREAM_TDEST	out	1	axis	VIDEO_OUT_V_dest_V	pointer
OUTPUT_STREAM_TVALID	out	1	axis	VIDEO_OUT_V_dest_V	pointer
OUTPUT_STREAM_TREADY	in	1	axis	VIDEO_OUT_V_dest_V	pointer
ap_clk	in	1	ap_ctrl_hs	image_filter	return value
ap_rst_n	in	1	ap_ctrl_hs	image_filter	return value
ap_done	out	1	ap_ctrl_hs	image_filter	return value
ap_start	in	1	ap_ctrl_hs	image_filter	return value
ap_idle	out	1	ap_ctrl_hs	image_filter	return value
ap_ready	out	1	ap_ctrl_hs	image_filter	return value

Implementation Results

- .

General Information

Report date:	Fri Oct 16 17:09:25 -0400 2015
Device target:	xc7z020clg484-1
Implementation tool:	Xilinx Vivado v.2015.3

Resource Usage

	Verilog
SLICE	1920
LUT	3137
FF	4908
DSP	150
BRAM	24
SRL	2

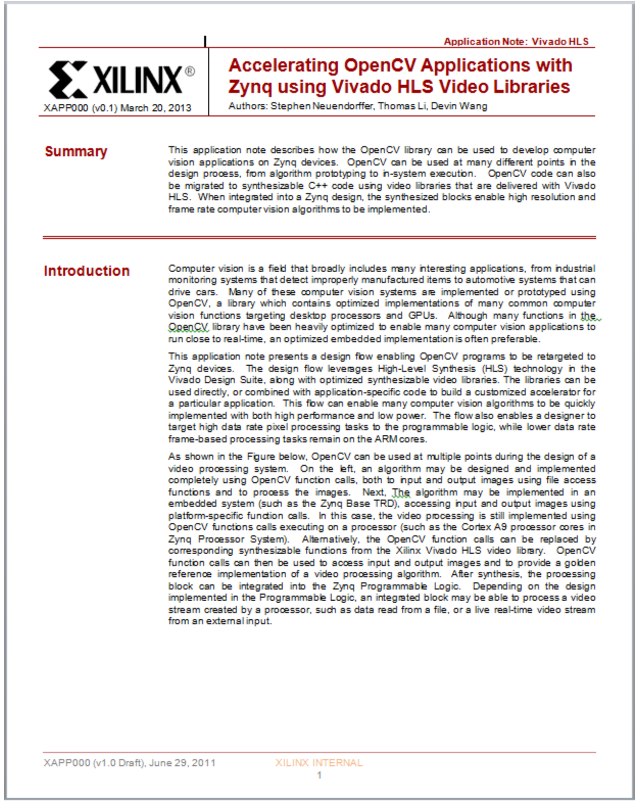
Final Timing

	Verilog
CP required	6.000
CP achieved	5.713

Timing met

Application Note XAPP1167

Accelerating OpenCV Applications with Zynq using Vivado HLS Video Libraries


Application Note: Vivado HLS

XILINX®

XAPP000 (v0.1) March 20, 2013

Accelerating OpenCV Applications with Zynq using Vivado HLS Video Libraries

Authors: Stephen Neuendorffer, Thomas Li, Devin Wang

Summary

This application note describes how the OpenCV library can be used to develop computer vision applications on Zynq devices. OpenCV can be used at many different points in the design process, from algorithm prototyping to in-system execution. OpenCV code can also be migrated to synthesizable C++ code using video libraries that are delivered with Vivado HLS. When integrated into a Zynq design, the synthesized blocks enable high resolution and frame rate computer vision algorithms to be implemented.

Introduction

Computer vision is a field that broadly includes many interesting applications, from industrial monitoring systems that detect improperly manufactured items to automotive systems that can drive cars. Many of these computer vision systems are implemented or prototyped using OpenCV, a library which contains optimized implementations of many common computer vision functions targeting desktop processors and GPUs. Although many functions in the OpenCV library have been heavily optimized to enable many computer vision applications to run close to real-time, an optimized embedded implementation is often preferable.

This application note presents a design flow enabling OpenCV programs to be retargeted to Zynq devices. The design flow leverages High-Level Synthesis (HLS) technology in the Vivado Design Suite, along with optimized synthesizable video libraries. The libraries can be used directly, or combined with application-specific code to build a customized accelerator for a particular application. This flow can enable many computer vision algorithms to be quickly implemented with both high performance and low power. The flow also enables a designer to target high data rate pixel processing tasks to the programmable logic, while lower data rate frame-based processing tasks remain on the ARM cores.

As shown in the Figure below, OpenCV can be used at multiple points during the design of a video processing system. On the left, an algorithm may be designed and implemented completely using OpenCV function calls, both to input and output images using file access functions and to process the images. Next, The algorithm may be implemented in an embedded system (such as the Zynq Base TRD), accessing input and output images using platform-specific function calls. In this case, the video processing is still implemented using OpenCV functions calls executing on a processor (such as the Cortex A9 processor cores in Zynq Processor System). Alternatively, the OpenCV function calls can be replaced by corresponding synthesizable functions from the Xilinx Vivado HLS video library. OpenCV function calls can then be used to access input and output images and to provide a golden reference implementation of a video processing algorithm. After synthesis, the processing block can be integrated into the Zynq Programmable Logic. Depending on the design implemented in the Programmable Logic, an integrated block may be able to process a video stream created by a processor, such as data read from a file, or a live real-time video stream from an external input.

XAPP000 (v1.0 Draft), June 29, 2011 XILINX INTERNAL 1

- **Video Processing data types**
- **Compares Video Architectures**
- **Advantages of Video Streaming**
- **Review Video Interfaces**
- **Reference Design with source files and project directories**

Download XAPP1167 from Xilinx.com