XILINX
ALL PROGRAMMABLE™

OpenCV on Zynq: Accelerating 4k60 Dense Optical Flow and Stereo Vision

정웅 부장, DSP Specialist
Nov 21, 2017

reVISION Stack

OpenVX Vision Acceleration | Caffe

Frameworks

OpenCV | SDSoC Environment | DNN, CNN, GoogLeNet, SSD, FCN …

Libraries and Tools




Development Kits

Agenda

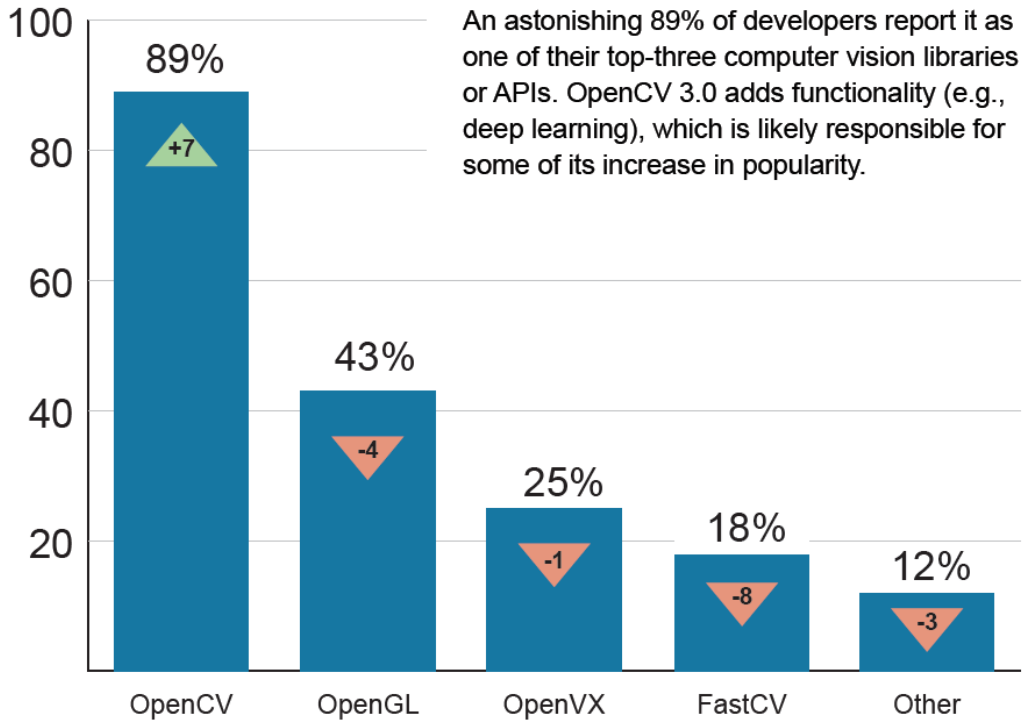
- Why Zynq SoCs for Traditional Computer Vision
- Automated Flow for OpenCV HW Acceleration
- Case Study
- What's available now for OpenCV of Revision Stack
- OpenCV and DNN live Demo with Revision

Why OpenCV with Xilinx

OpenCV Needs Acceleration in Embedded

- Typical ARM Cortex-A53

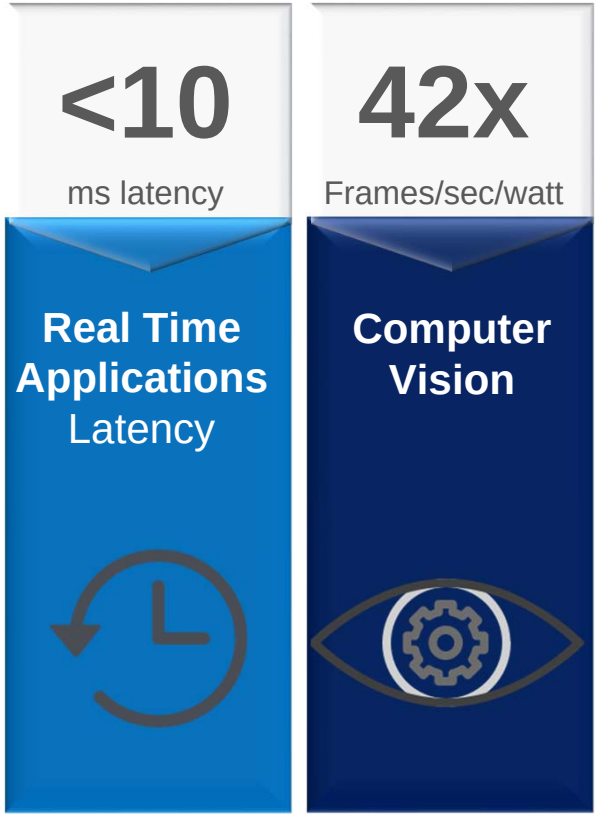
Typical Requirement	> 30 FPS
Harris Corner	2.4 FPS
Stereo Depth Map	2.1 FPS
Dense Optical Flow	0.1 FPS



OpenCV dominates development.
An astonishing 89% of developers report it as one of their top-three computer vision libraries or APIs. OpenCV 3.0 adds functionality (e.g., deep learning), which is likely responsible for some of its increase in popularity.

Source: Embedded Vision Alliance,
Embedded Vision Developer Survey, January 2017

Zynq Offer Superior Performance, Latency

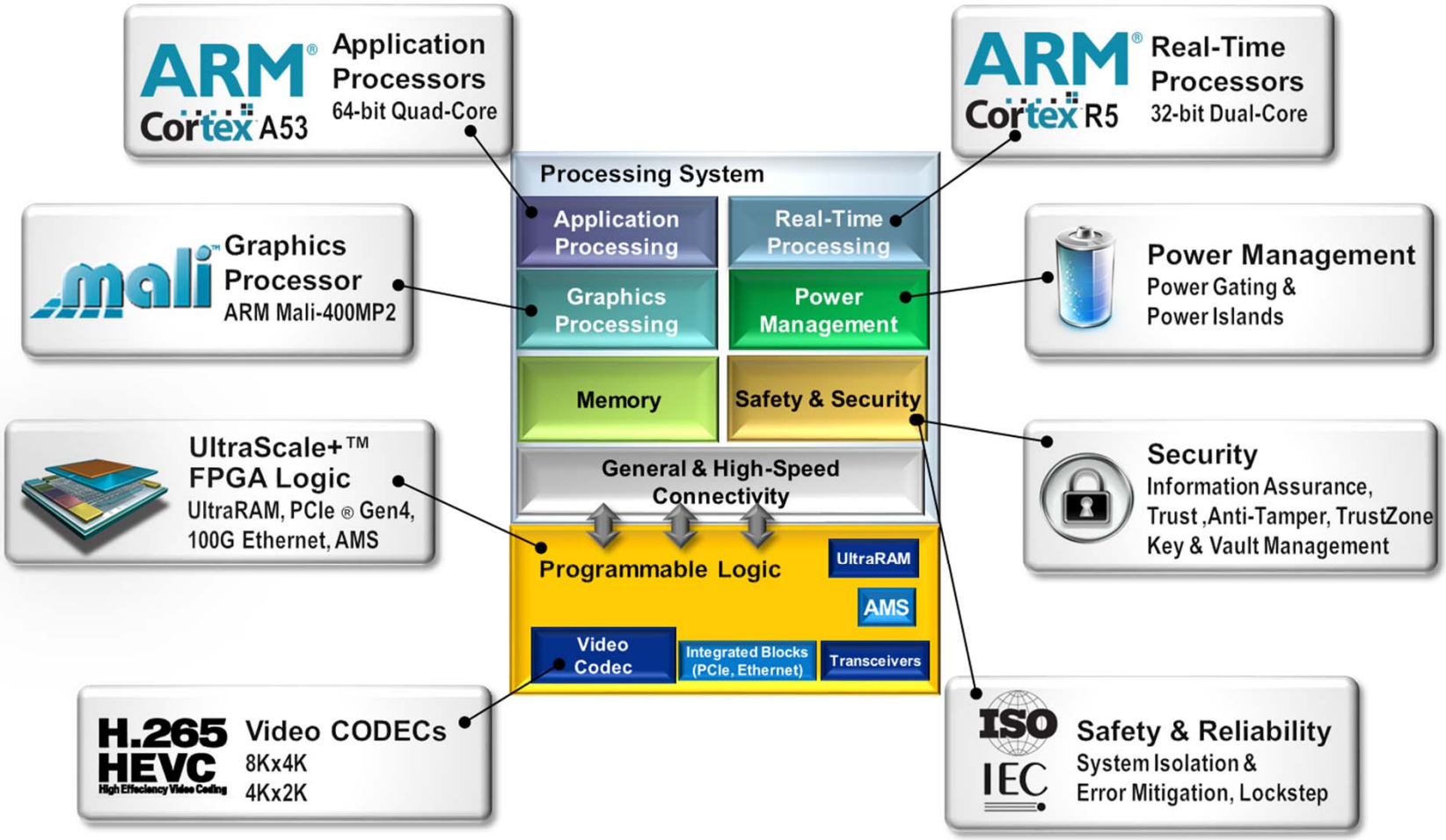


CV:: StereoLBM @1080p		Xilinx ZU9	Xilinx ZU5	eGPU*
	Frames/s	**700**	**296**	43
	Power (W)	**4.8**	**3.3**	7.9
	Frames/s/watt	**145.8**	**89.7**	5.4

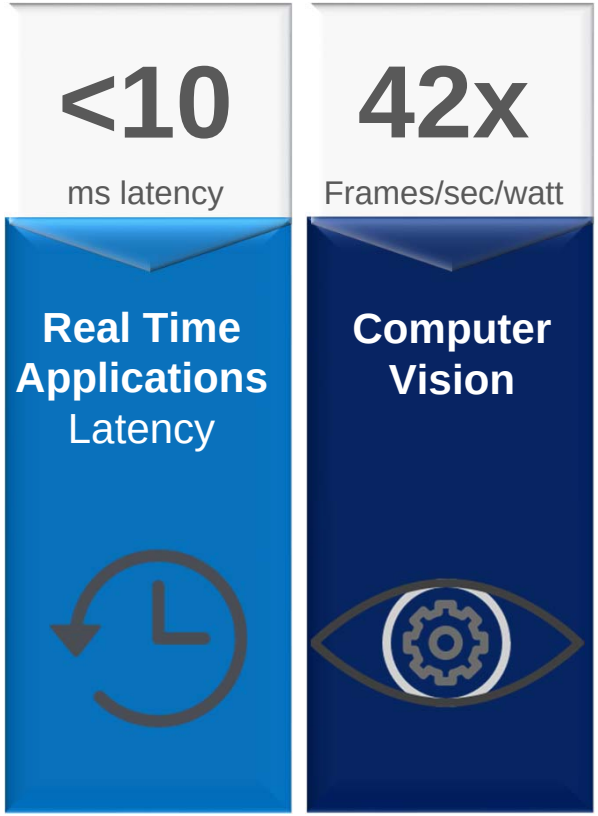
CV:: LK Dense Optical Flow @720p		Xilinx ZU9	Xilinx ZU5	eGPU*
	Frames/s	**170**	**73**	7
	Power (W)	**4.8**	**3.3**	7.9
	Frames/s/watt	**35.4**	**22.1**	0.9

- eGPU = nVidia Tegra X1 using VisionWorks for StereoLBM and OpenCV4Tegra for OpticalFlow
- All benchmarks utilize as much resources as possible on GPU (~99%) and programmable logic (~70%)

Zynq Offers the Most Efficient CV Acceleration



Zynq Offer Superior Performance, Latency

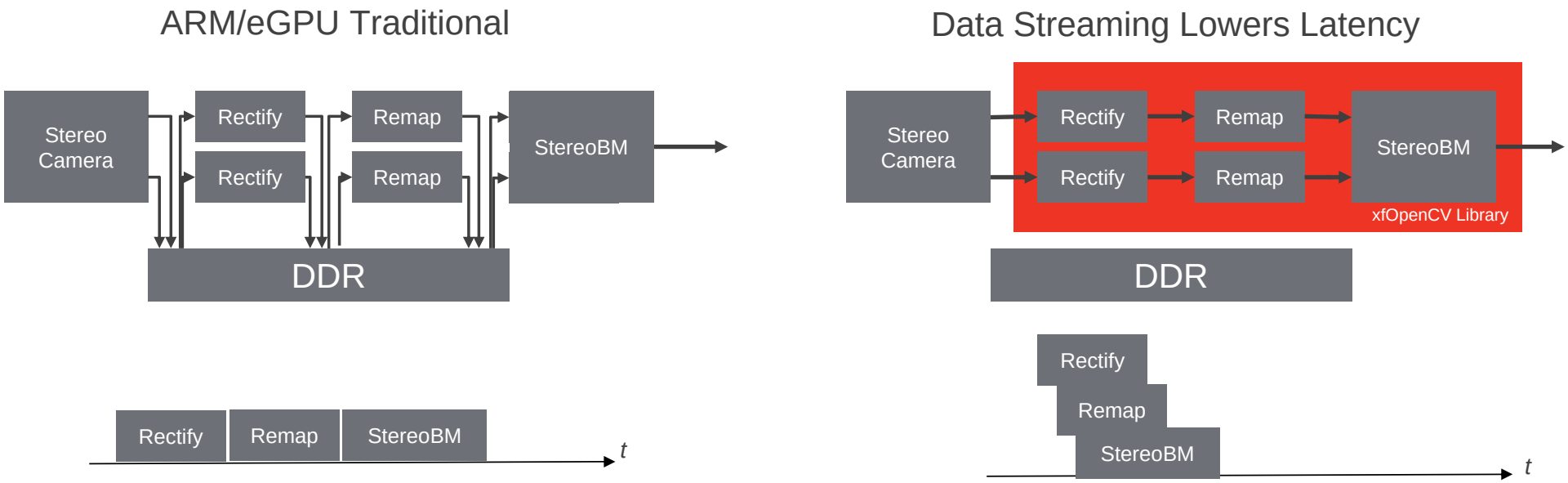


CV:: StereoLBM @1080p		Xilinx ZU9	Xilinx ZU5	eGPU*
	Frames/s	**700**	**296**	43
	Power (W)	**4.8**	**3.3**	7.9
	Frames/s/watt	**145.8**	**89.7**	5.4

CV:: LK Dense Optical Flow @720p		Xilinx ZU9	Xilinx ZU5	eGPU*
	Frames/s	**170**	**73**	7
	Power (W)	**4.8**	**3.3**	7.9
	Frames/s/watt	**35.4**	**22.1**	0.9

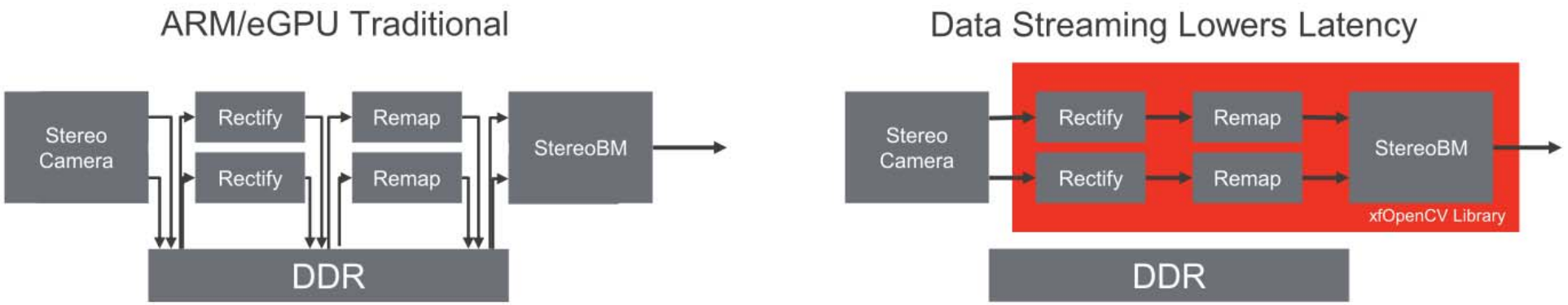
- eGPU = nVidia Tegra X1 using VisionWorks for StereoLBM and OpenCV4Tegra for OpticalFlow
- All benchmarks utilize as much resources as possible on GPU (~99%) and programmable logic (~70%)

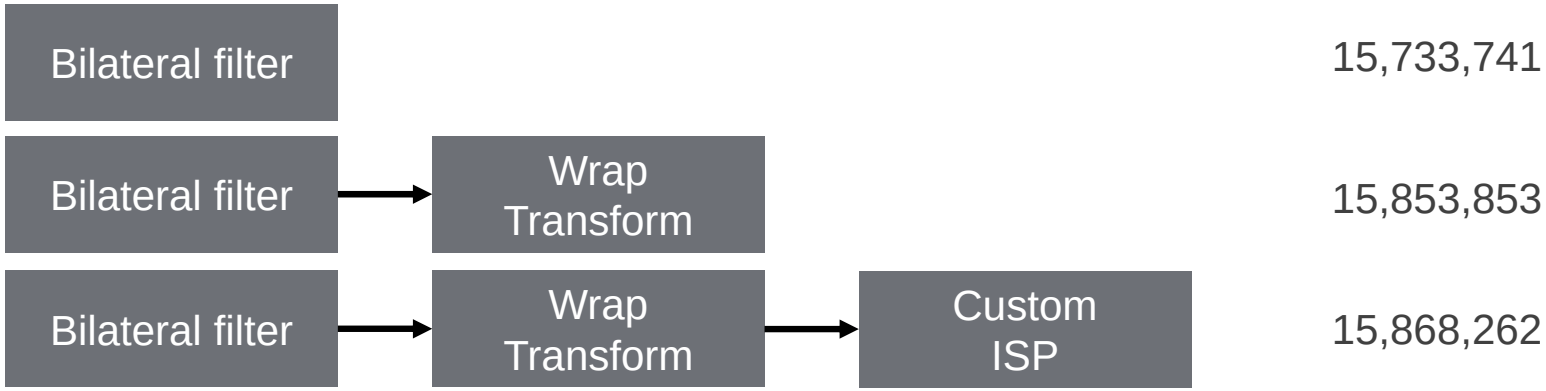
Why So Good? Efficient Window-based Streaming CV increases Performance



- xfOpenCV directly infers pipelining functions from one to the next, avoiding frame buffers and external memory

Streaming CV increases Performance





- xfOpenCV directly infers pipelining functions from one to the next, avoiding frame buffers and external memory

Desktop OpenCV to Zynq

OpenCV Support with Automatic HW Acceleration

1. Cross-compile OpenCV application to Zynq (ARM A9/A53)
2. Profile and identify bottleneck functions
3. Minimal changes to the code and set functions to hardware. Compile using SDSoC
4. Run on a Zynq board

OpenCV

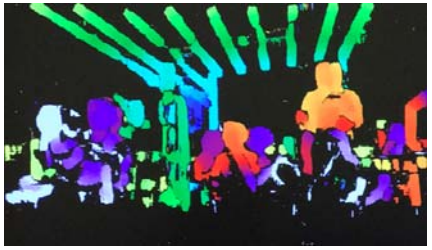
```
main(){
cv::imread(A);
cv::stereoRectify(A,B,C,D);
cv::stereoLBM(C,D,out);
cv::imshow(out);
}
```

Time [%]	Time
	4504 ms (100 %)
	4500 ms (100 %)
	4370 ms (97,1 %)
	4370 ms (97,1 %)
	3474 ms (77,2 %)
	3473 ms (77,2 %)
	3471 ms (77,2 %)
	3412 ms (75,8 %)
	2597 ms (57,7 %)
	2274 ms (50,5 %)
	1933 ms (43 %)
	1895 ms (42,1 %)
	1782 ms (39,6 %)
	1593 ms (35,4 %)
	1567 ms (34,8 %)
	1444 ms (32,1 %)

HW functions

Name	Clock Frequency (MHz)
stereoRectify	300
stereoLBM	300

```
main(){
cv::imread(A);
xF:stereoRectify<line>(A,B,C,D);
xF:stereoLBM<win,n_disp>(C,D,out);
cv::imshow(out);
}
```



XILINX ALL PROGRAMMABLE

xfOpenCV: 50+ Most Needed OpenCV Functions

Basic Functionality	Geometric Transforms	Image Processing and Filters	Feature Detection and Classifiers	3D Reconstruction	Motion Analysis and Tracking
Absolute difference	Scale/Resize	Box	Canny edge detection	StereoLBM	Mean Shift Tracking (MST)
Accumulate	StereoRectify	Gaussian	Fast corner		LK Dense Optical Flow
Accumulate squared	Warp Affine	Median	SVM (binary)		
Accumulate weighted	Warp Perspective	Sobel	Harris corner		
Arithmetic addition	Remap	Custom convolution	Histogram of Oriented Gradients (HOG)		
Arithmetic subtraction		Equalize Histogram			
Bitwise: AND, OR, XOR, NOT		Dilate			
Pixel-wise multiplication		Erode			
Channel combine		Bilateral			
Channel extract		OTSU Thresholding			
Color convert		Thresholding			
Convert bit depth		Image pyramid			
Table lookup		Color Detection			
		Integral image			
		Gradient Magnitude			
		Histogram			
		Gradient Phase			
		Min/Max Location			
		Mean & Standard Deviation			

xfOpenCV: Fast and Simple

Standard OpenCV Bilateral filter example

```
using namespace cv;

Mat imgOutput,
Mat imgInput,

// Step 1: Read in images
imgInput = imread(argv[1], 0);


// Step 2: Call Bilateral function
bilateralFilter(imgInput, imgOutput, 3, sigma_color,
sigma_space, cv::BORDER_REPLICATE);

// Step 3: Read out image
imwrite("output_ocv.png", imgOutput);
```

xfOpencv Bilateral filter example

```
using namespace xf;

Mat<XF_8UC1, HEIGHT, WIDTH, XF_NPPC1> imgInput;
Mat<XF_8UC1, HEIGHT, WIDTH, XF_NPPC1> imgOutput;

// Step 1: Read in images
imgInput = imread(argv[1], 0);


// Step 2: Call Bilateral function
bilateralFilter<3, XF_BORDER_REPLICATE,XF_8UC1, HEIGHT, WIDTH,
XF_NPPC1>(imgInput, imgOutput, sigma_color, sigma_space);

// Step 3: read out image
imwrite("output_ocv.png", imgOutput);
```

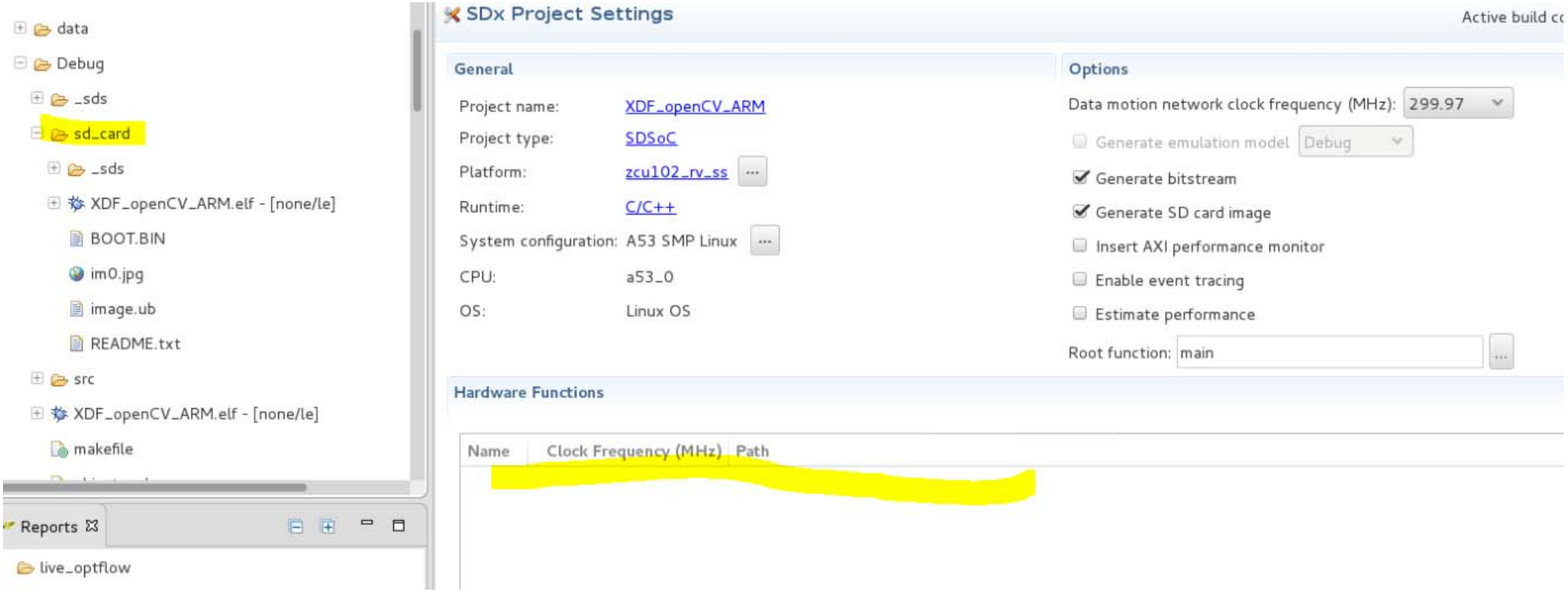
Step 1: Standard OpenCV on ARM

Standard OpenCV Bilateral filter example

```
using namespace cv;

Mat imgOutput,
Mat imgInput,

// Step 1: Read in images
imgInput = imread(argv[1], 0);


// Step 2: Call Bilateral function
bilateralFilter(imgInput, imgOutput, 3, sigma_color,
sigma_space, cv::BORDER_REPLICATE);

// Step 3: Read out image
imwrite("output_ocv.png", imgOutput);
```

- **Simply import the C/C++ projects with OpenCV APIs into SDSoC**
- **All necessary OpenCV compile / linking environments for ARM are provided**
- **Ready-to-compile!**

Step 1: Build SDCard image with nothing in hardware

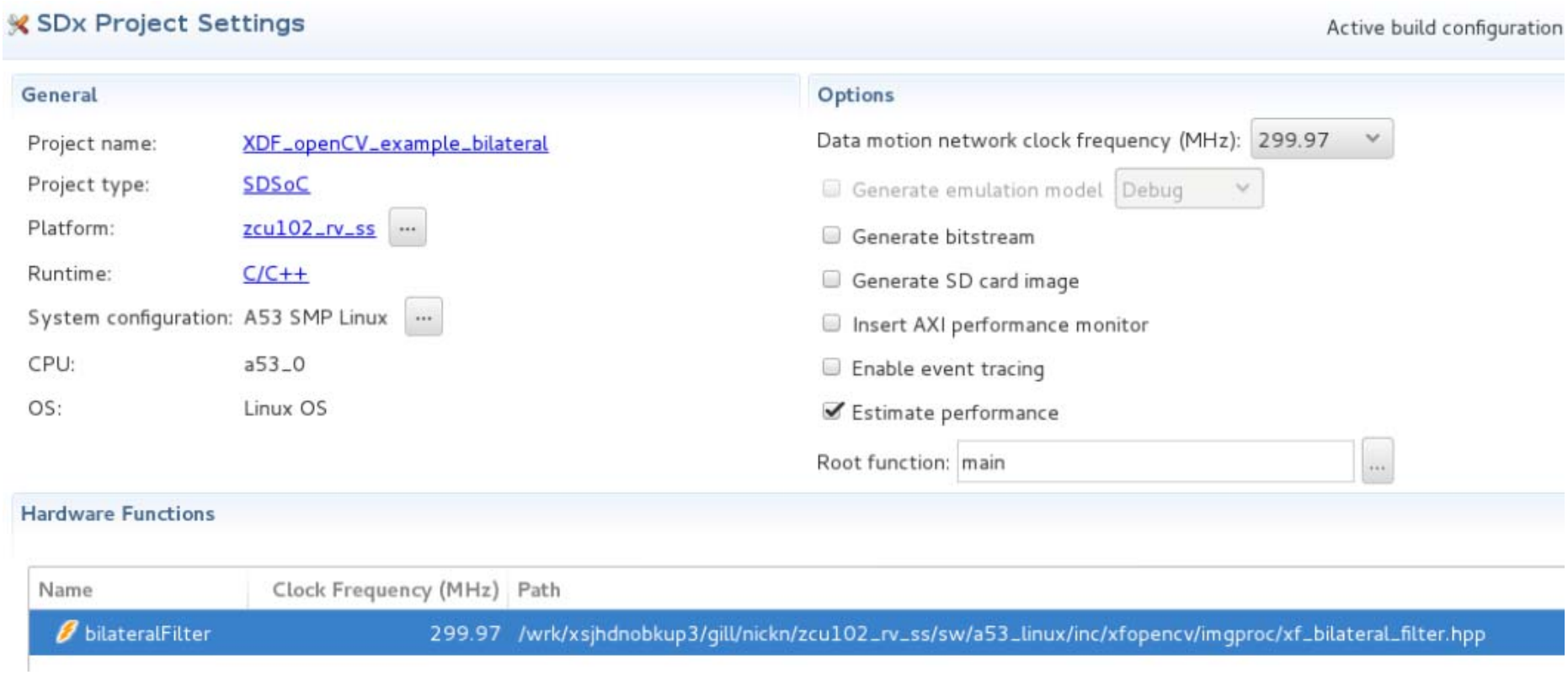


Step 2: Move bilateralFilter to hardware

- Minor mods needed to use OpenCV libraries for hardware acceleration
  - Namespace change: “cv::” to “xf::”
  - Add template parameters for optimized hardware generation

```
using namespace xf;

Mat<XF_8UC1, HEIGHT, WIDTH, XF_NPPC1> imgInput;
Mat<XF_8UC1, HEIGHT, WIDTH, XF_NPPC1> imgOutput;

// Step 1: Read in images
imgInput = imread(argv[1], 0);


// Step 2: Call Bilateral function
bilateralFilter<3, XF_BORDER_REPLICATE,XF_8UC1,
HEIGHT, WIDTH, XF_NPPC1>(imgInput, imgOutput,
sigma_color, sigma_space);

// Step 3: read out image
imwrite("output_ocv.png", imgOutput);
```

Step 2: Assign to hardware



Step 2: Estimate Performance and Build

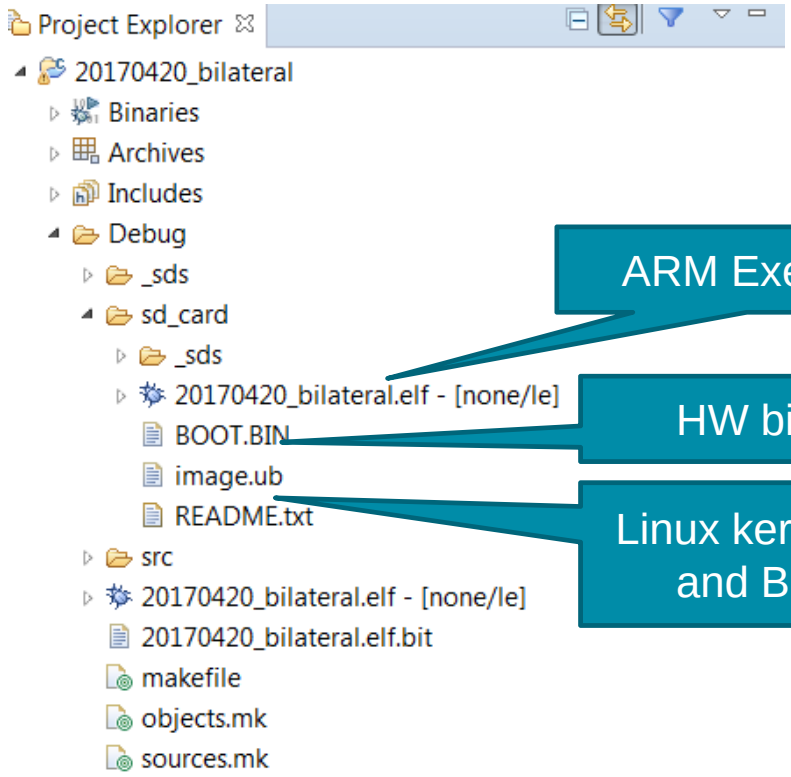
- Fast estimation in minutes to get system-level performance and HW utilization
- Build the full system with a click of button

Performance estimates for 'wO_xf_bilateralFilter in xf_bi ...

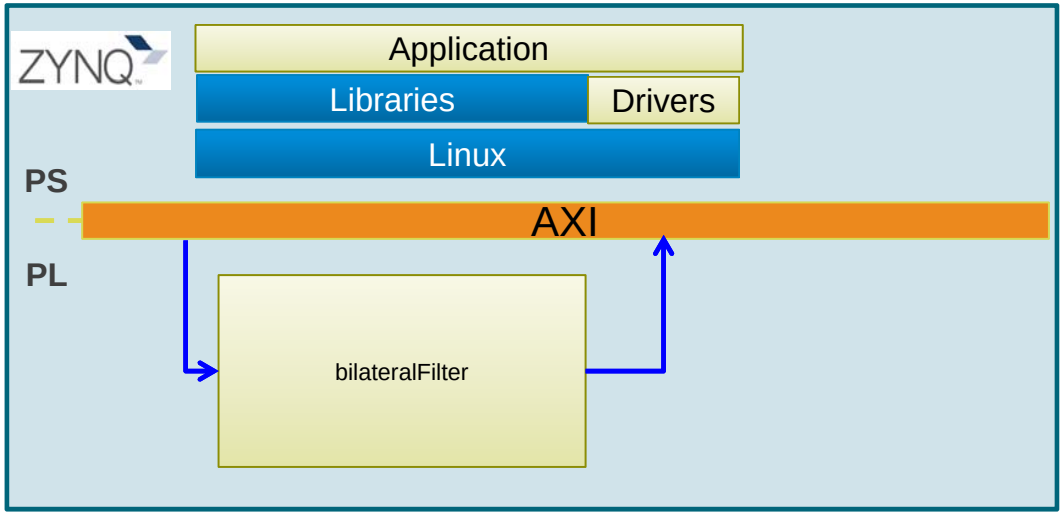
Hardware accelerated (Estimated cycl	15733741

Resource utilization estimates for Hardware functions

Resource	Used	Total	% Utilization
DSP	21	2520	0.83
BRAM	11	912	1.21
LUT	13774	274080	5.03
FF	14764	548160	2.69



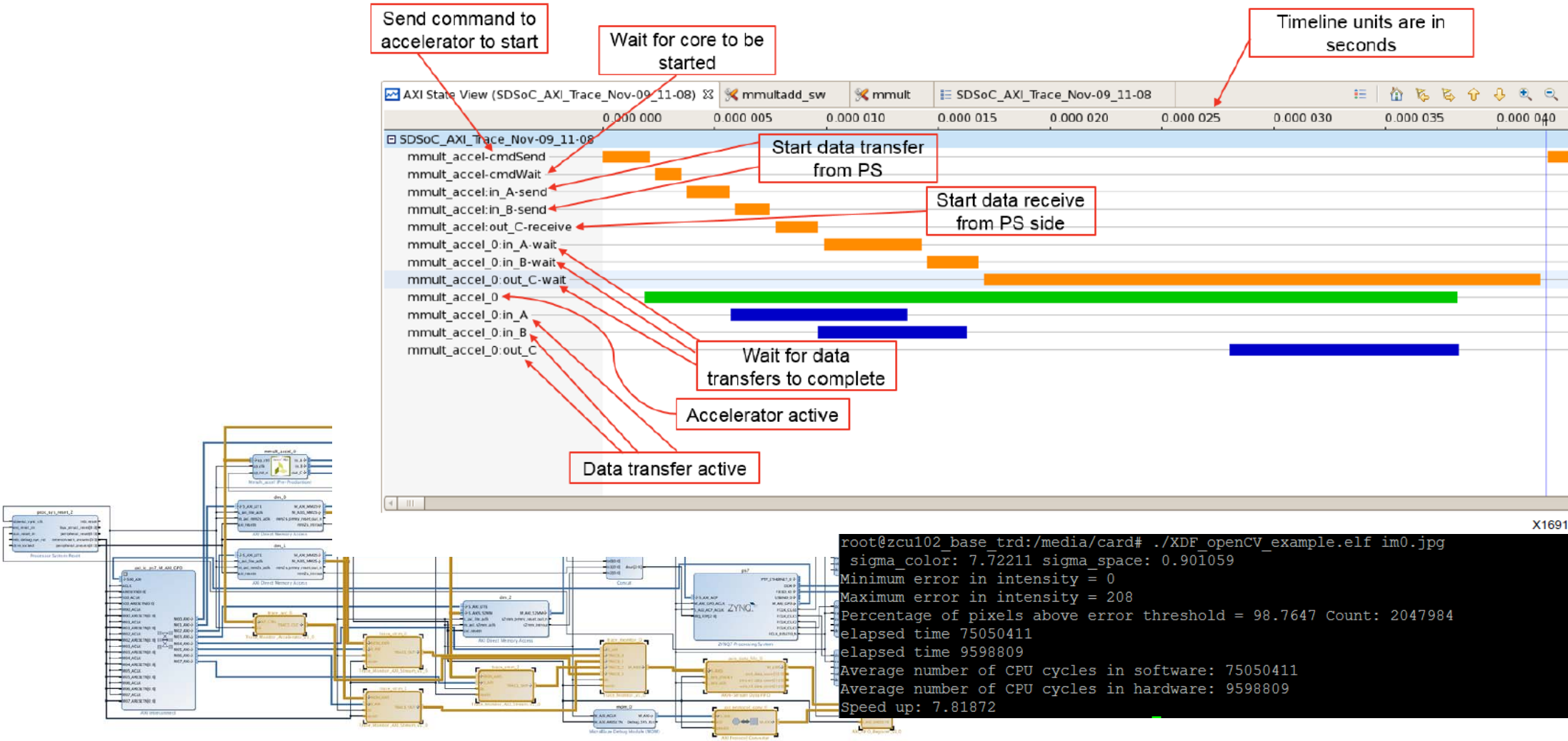
Step 2: Data motion report



Data Motion Network

Accelerator	Argument	IP Port	Direction	Declared Size(bytes)	Pragmas	Connection
w0_xf_bilateralFilter_1	_src_mat	p_src_mat_rows	IN	4		axi_interconnect_hpm1_M08_AXI:AXILITE:0xC
		p_src_mat_cols	IN	4		axi_interconnect_hpm1_M08_AXI:AXILITE:0x10
		p_src_mat_size	IN	4		axi_interconnect_hpm1_M08_AXI:AXILITE:0x14
		p_src_mat_data_V	IN	2073600*1	• length:(_src_mat.size) • mem_attribute:NON_CACHEABLE.PHYSICAL_CONTIGUOUS	zynq_ultra_ps_e_0_S_AXI_HP3_FPD:AXIDMA_SG
	_dst_mat	p_dst_mat_rows	IN	4		axi_interconnect_hpm1_M08_AXI:AXILITE:0x18
		p_dst_mat_cols	IN	4		axi_interconnect_hpm1_M08_AXI:AXILITE:0x1C
		p_dst_mat_size	IN	4		axi_interconnect_hpm1_M08_AXI:AXILITE:0x20
		p_dst_mat_data_V	OUT	2073600*1	• length:(_dst_mat.size) • mem_attribute:NON_CACHEABLE.PHYSICAL_CONTIGUOUS	zynq_ultra_ps_e_0_S_AXI_HP3_FPD:AXIDMA_SG
	sigma_space	sigma_space	IN	4		axi_interconnect_hpm1_M08_AXI:AXILITE:0x24
	sigma_color	sigma_color	IN	4		axi_interconnect_hpm1_M08_AXI:AXILITE:0x28

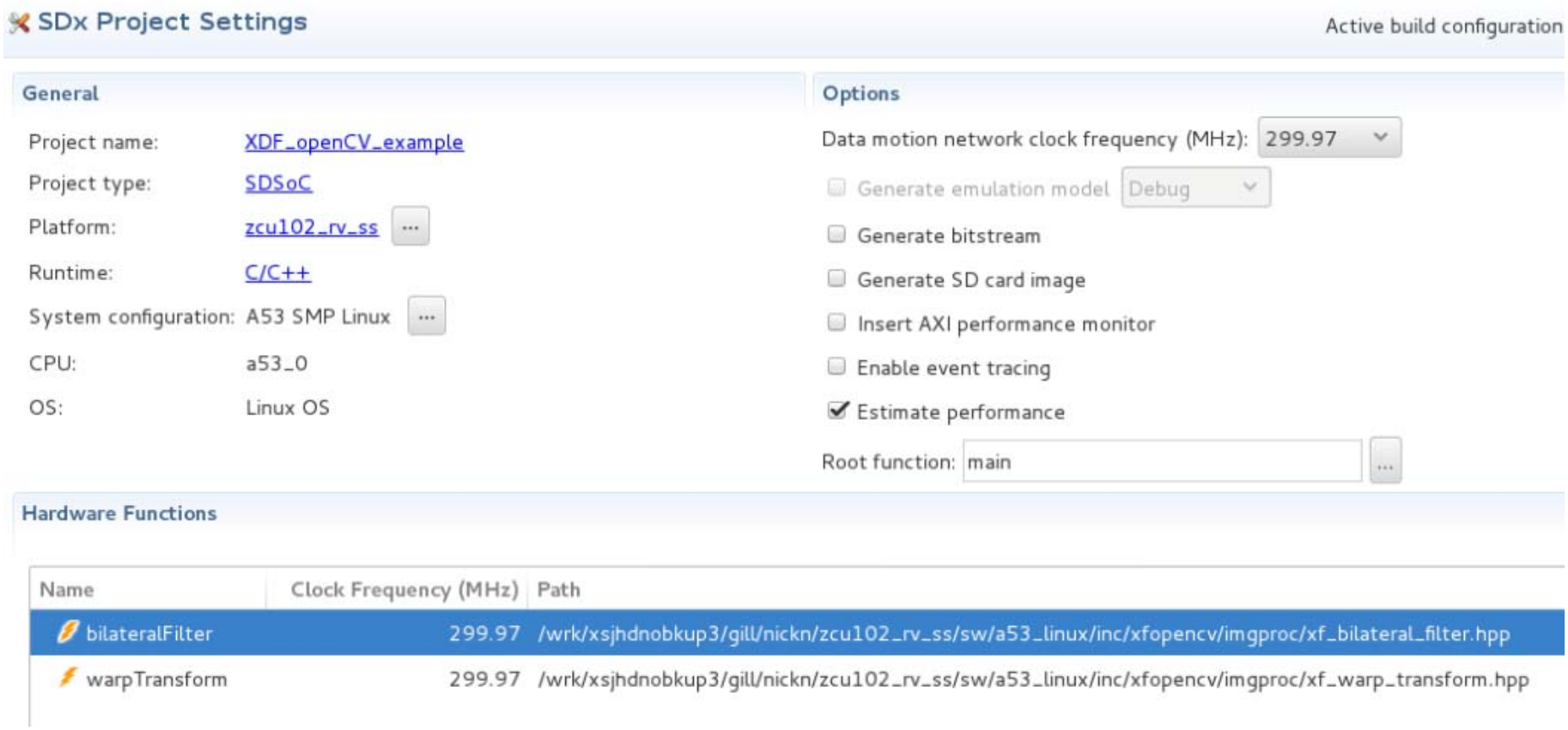
Step 2: Run on a Board and Collect Traces



Step 3: Add warpTransform to hardware

```
using namespace xf;

Mat<XF_8UC1, HEIGHT, WIDTH, XF_NPPC1> imgInput;
Mat<XF_8UC1, HEIGHT, WIDTH, XF_NPPC1> imgOutput;
Mat<XF_8UC1, HEIGHT, WIDTH, XF_NPPC1> imgOutput2;

// Step 1: Read in images
imgInput = imread(argv[1], 0);


// Step 2: Call Bilateral & WarpTransform function
bilateralFilter<3, XF_BORDER_REPLICATE,XF_8UC1, HEIGHT, WIDTH, XF_NPPC1>(imgInput,
imgOutput, sigma_color, sigma_space);
warpTransform<NUM_STORE_ROWS, START_PROC, TRANSFORM_TYPE, INTERPOLATION, XF_8UC1,
HEIGHT, WIDTH, XF_NPPC1>(ImgOutput, ImgOutput2, R);

// Step 3: read out image
imwrite("output_ocv.png", imgOutput2);
```

Step 3: Assign to hardware



Step 3: Estimate Performance and Build

- Notice the latency almost did NOT change

Details

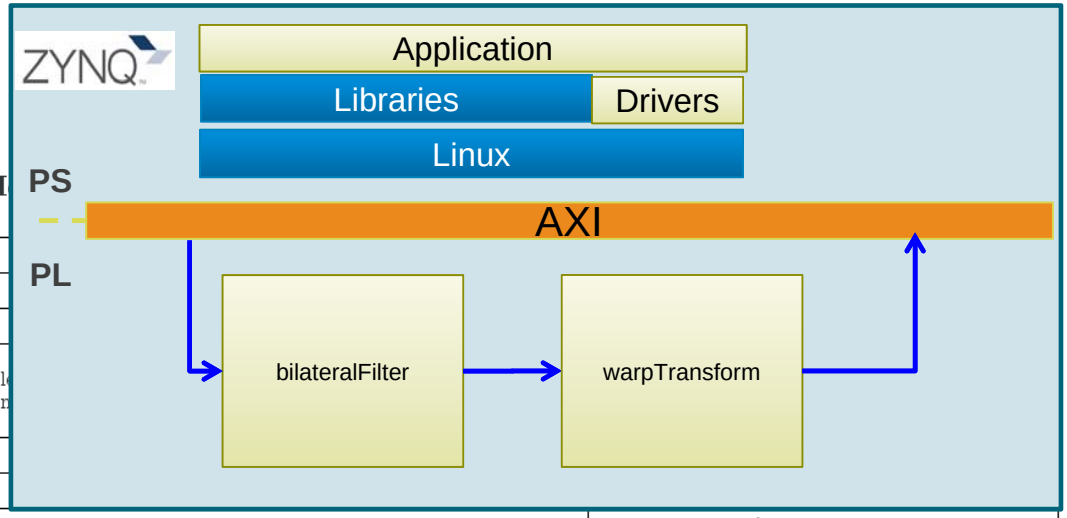
Performance estimates for functions 'wO_xf_bilateralFilte ...

Hardware accelerated (Estimated cycl	15853853

Resource utilization estimates for Hardware functions

Resource	Used	Total	% Utilization
DSP	61	2520	2.42
BRAM	69	912	7.57
LUT	29876	274080	10.9
FF	29477	548160	5.38

Step 3: Data motion report



Data M

Accelerator	Argument	IP Port	Direction	Declared Size(bytes)		
w0_xf_bilateralFilter_1	_src_mat	p_src_mat_rows	IN	4		
		p_src_mat_cols	IN	4		
		p_src_mat_size	IN	4		
		p_src_mat_data_V	IN	2073600*1	• l • n	
	_dst_mat	p_dst_mat_rows	IN	4		
		p_dst_mat_cols	IN	4		
		p_dst_mat_size	IN	4		axi_interconnect_hpm1_M08_AXI:AXILITE:0x20
		p_dst_mat_data_V	OUT	2073600*1	• length:(_dst_mat.size) • mem_attribute:NON_CACHEABLE.PHYSICAL_CONTIGUOUS	w1_xf_warpTransform_1:p_src_mat_data_V
	sigma_space	sigma_space	IN	4		axi_interconnect_hpm1_M08_AXI:AXILITE:0x24
	sigma_color	sigma_color	IN	4		axi_interconnect_hpm1_M08_AXI:AXILITE:0x28
w1_xf_warpTransform_1	_src_mat	p_src_mat_rows	IN	4		axi_interconnect_hpm1_M08_AXI:AXILITE:0xC
		p_src_mat_cols	IN	4		axi_interconnect_hpm1_M08_AXI:AXILITE:0x10
		p_src_mat_size	IN	4		axi_interconnect_hpm1_M08_AXI:AXILITE:0x14
		p_src_mat_data_V	IN	2073600*1	• length:(_src_mat.size) • mem_attribute:NON_CACHEABLE.PHYSICAL_CONTIGUOUS	w0_xf_bilateralFilter_1:p_dst_mat_data_V
	_dst_mat	p_dst_mat_rows	IN	4		axi_interconnect_hpm1_M08_AXI:AXILITE:0x18
		p_dst_mat_cols	IN	4		axi_interconnect_hpm1_M08_AXI:AXILITE:0x1C
		p_dst_mat_size	IN	4		axi_interconnect_hpm1_M08_AXI:AXILITE:0x20
		p_dst_mat_data_V	OUT	2073600*1	• length:(_dst_mat.size) • mem_attribute:NON_CACHEABLE.PHYSICAL_CONTIGUOUS	zynq_ultra_ps_e_0_S_AXI_HP3_FPD:AXIDMA_SG
	P_matrix	P_matrix_PORTA	IN	9*4		axi_interconnect_hpm1_M08_AXI:AXIFIFO

Add User CV Library for Acceleration

Custom CV Function / Library Creation Flow

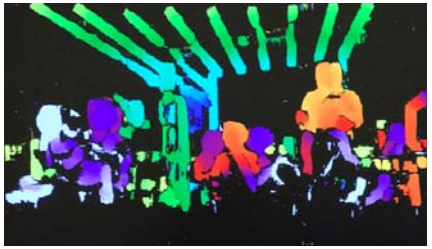
1. Write custom CV function in C, C++ or OpenCL. Cross-compile to Zynq (ARM A9/A53)
2. Optimize for hardware using SDSoC
3. Assign functions to hardware. Compile using SDSoC
4. Run on a Zynq board

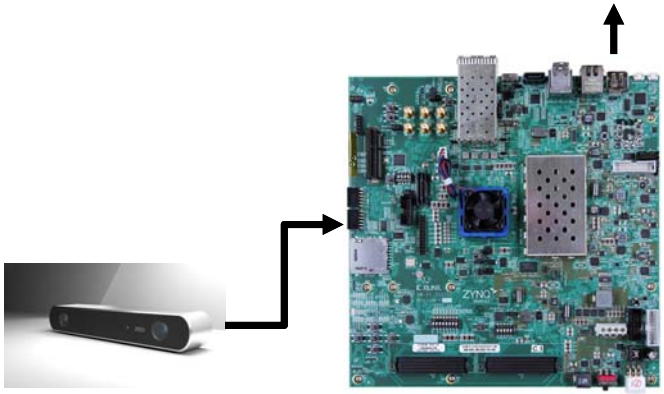
```
main(){
cv::imread(A);
xF:stereoRectify<line>(A,B,C,D);
xF:stereoLBM<win,n_disp>(C,D,E);
CUSTOM_CV(E,out);
cv::imshow(out);
}
```

```
CUSTOM_CV(E,out){
  #pragma HLS PIPELINE
  for(…){
   #pragma HLS UNROLL
   for(…){ …
   }
  }
}
```

HW functions

Name	Clock Frequency (MHz)
stereoRectify	300
stereoLBM	300
CUSTOM_CV	300



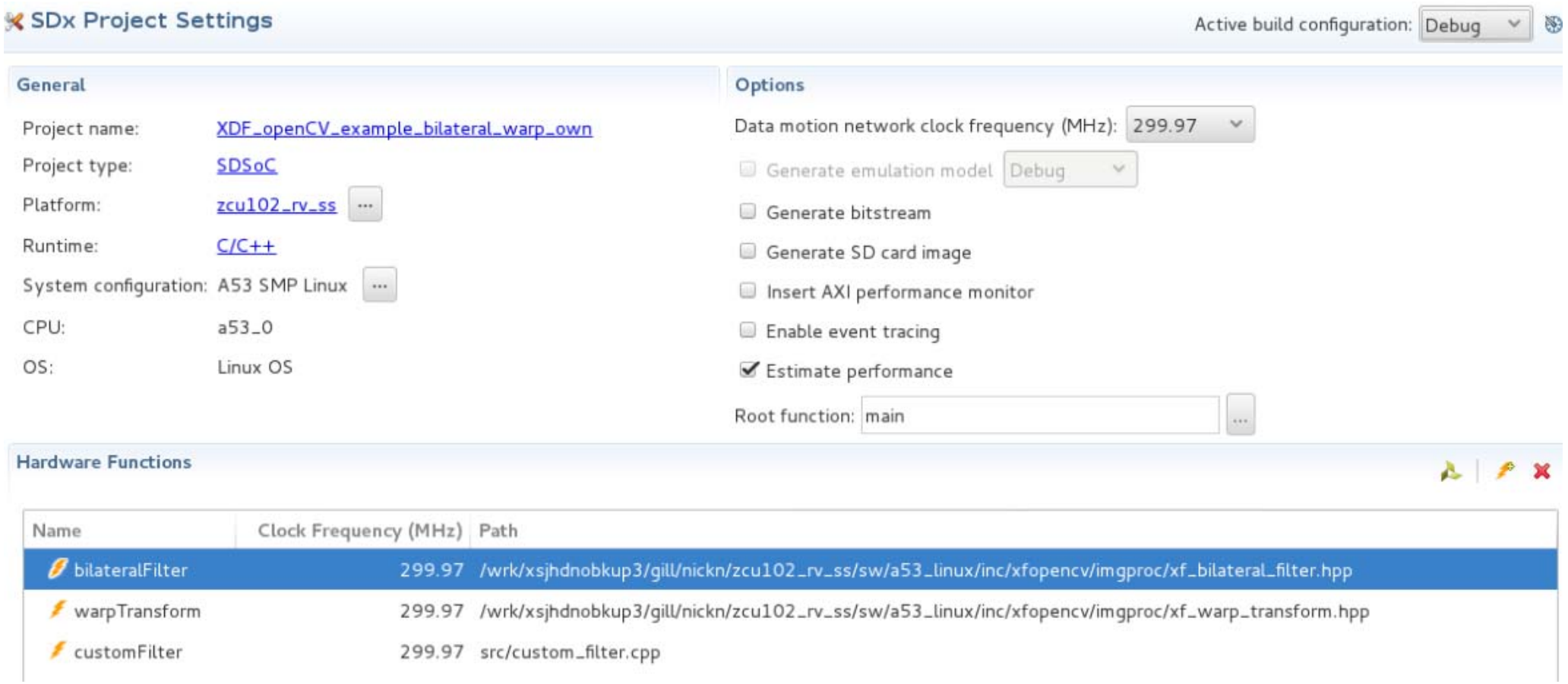


Step 4: Add custom function to hardware

```
using namespace xf;

// Step 2: Call Bilateral & WarpTransform function
bilateralFilter<3, XF_BORDER_REPLICATE,XF_8UC1, HEIGHT, WIDTH, XF_NPPC1>(imgInput,
imgOutput, sigma_color, sigma_space);
warpTransform<NUM_STORE_ROWS, START_PROC, TRANSFORM_TYPE, INTERPOLATION, XF_8UC1, HEIGHT,
WIDTH, XF_NPPC1>(ImgOutput, ImgOutput2, R);
customFilter(ImgOutput2, ImgOutput3);
```

```
void customFilter(xf::Mat<XF_8UC1, 1080, 1920, XF_NPPC1> & _src_mat,xf::Mat<XF_8UC1, 1080, 1920, XF_NPPC1> & _dst_mat
{
    for(int i=0; i<_src_mat.rows;i++)
    {
        for(int j=0; j<(_src_mat.cols)>>(XF_BITSHIFT(XF_NPPC1));j++)
        {
#pragma HLS PIPELINE
            *(_dst_mat.data + i*(_dst_mat.cols>>(XF_BITSHIFT(XF_NPPC1))) +j) = *(_src_mat.data + i*(_src_mat.cols>>(X
        }
    }

}
```

Step 4: Assign to hardware



Step 4: Estimate Performance and Build

- Notice the latency almost did NOT change… again

Performance estimates for functions 'w1_xf_bilateralFilte ...

Hardware accelerated (Estimated cycl	15868262

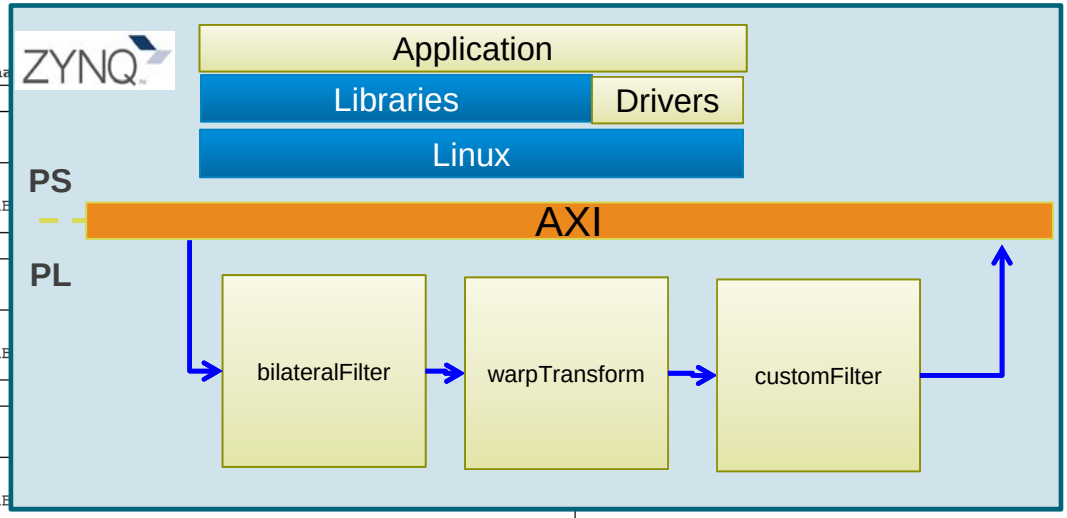
Resource utilization estimates for Hardware functions

Resource	Used	Total	% Utilization
DSP	61	2520	2.42
BRAM	69	912	7.57
LUT	30108	274080	10.99
FF	29580	548160	5.4

Step 4: Data motion report

Data Motion Network

Accelerator	Argument	IP Port	Direction	Declared Size(bytes)	Pragma	
customFilter_1	_src_mat	p_src_mat_rows	IN	4		
		p_src_mat_cols	IN	4		
		p_src_mat_size	IN	4		
		p_src_mat_data_V	IN	2073600*1	• length:(_src_mat.size) • mem_attribute:NON_CACHEAB	
	_dst_mat	p_dst_mat_rows	IN	4		
		p_dst_mat_cols	IN	4		
		p_dst_mat_size	IN	4		
		p_dst_mat_data_V	OUT	2073600*1	• length:(_dst_mat.size) • mem_attribute:NON_CACHEAB	
w1_xf_bilateralFilter_1	_src_mat	p_src_mat_rows	IN	4		
		p_src_mat_cols	IN	4		
		p_src_mat_size	IN	4		
		p_src_mat_data_V	IN	2073600*1	• length:(_src_mat.size) • mem_attribute:NON_CACHEAB	
	_dst_mat	p_dst_mat_rows	IN	4		axi_interconnect_hpm1_M08_AXI:AXILITE:0x18
		p_dst_mat_cols	IN	4		axi_interconnect_hpm1_M08_AXI:AXILITE:0x1C
		p_dst_mat_size	IN	4		axi_interconnect_hpm1_M08_AXI:AXILITE:0x20
		p_dst_mat_data_V	OUT	2073600*1	• length:(_dst_mat.size) • mem_attribute:NON_CACHEABLE.PHYSICAL_CONTIGUOUS	w2_xf_warpTransform_1:p_src_mat_data_V
	sigma_space	sigma_space	IN	4		axi_interconnect_hpm1_M08_AXI:AXILITE:0x24
	sigma_color	sigma_color	IN	4		axi_interconnect_hpm1_M08_AXI:AXILITE:0x28
w2_xf_warpTransform_1	_src_mat	p_src_mat_rows	IN	4		axi_interconnect_hpm1_M08_AXI:AXILITE:0xC
		p_src_mat_cols	IN	4		axi_interconnect_hpm1_M08_AXI:AXILITE:0x10
		p_src_mat_size	IN	4		axi_interconnect_hpm1_M08_AXI:AXILITE:0x14
		p_src_mat_data_V	IN	2073600*1	• length:(_src_mat.size) • mem_attribute:NON_CACHEABLE.PHYSICAL_CONTIGUOUS	w1_xf_bilateralFilter_1:p_dst_mat_data_V
	_dst_mat	p_dst_mat_rows	IN	4		axi_interconnect_hpm1_M08_AXI:AXILITE:0x18
		p_dst_mat_cols	IN	4		axi_interconnect_hpm1_M08_AXI:AXILITE:0x1C
		p_dst_mat_size	IN	4		axi_interconnect_hpm1_M08_AXI:AXILITE:0x20
		p_dst_mat_data_V	OUT	2073600*1	• length:(_dst_mat.size) • mem_attribute:NON_CACHEABLE.PHYSICAL_CONTIGUOUS	customFilter_1:p_src_mat_data_V
	P_matrix	P_matrix_PORTA	IN	9*4		axi_interconnect_hpm1_M08_AXI:AXIFIFO

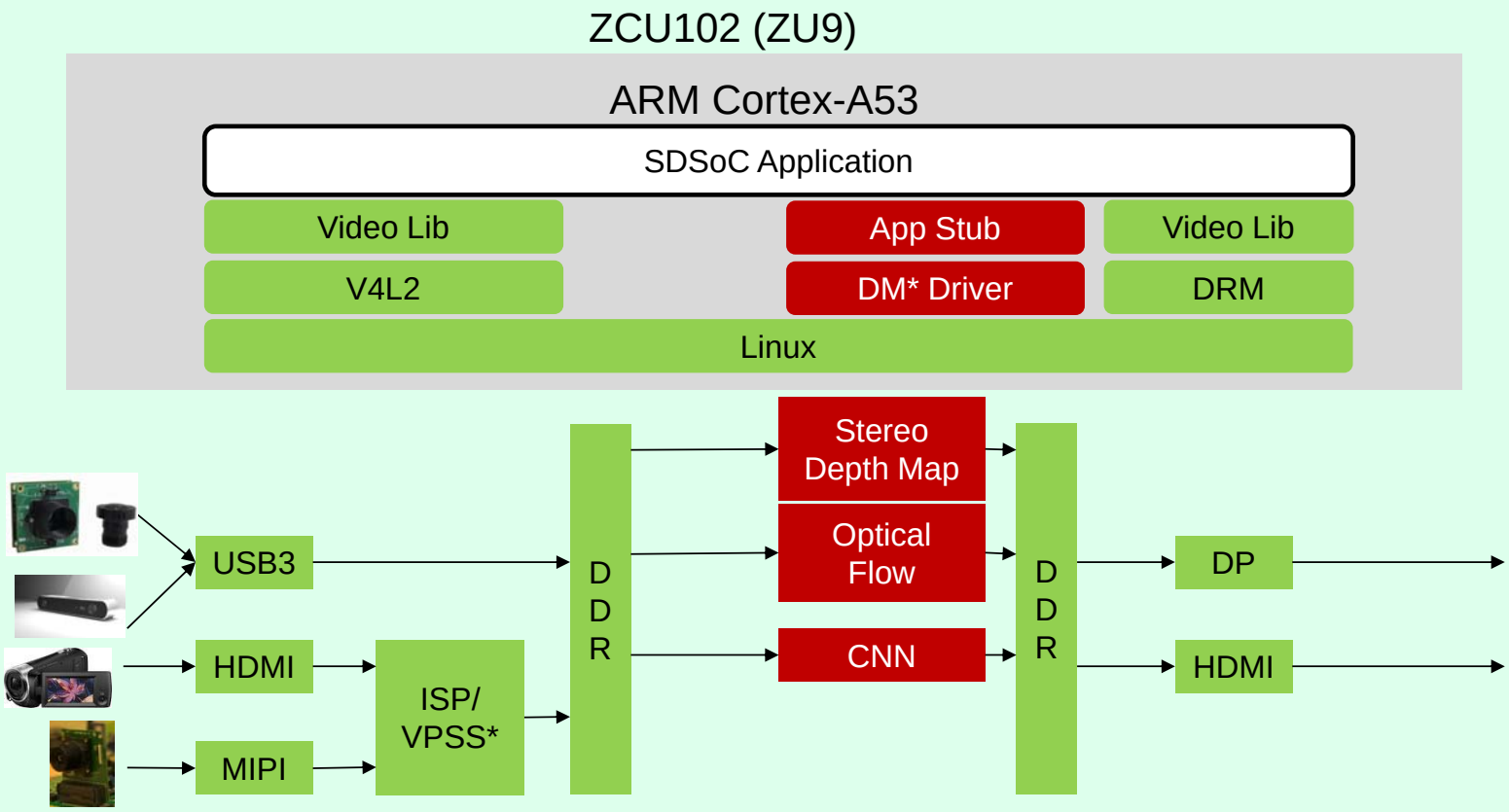


Case Study

Embedded Vision (reVISION) Platform

SDSoC Generated

SDSoC Platform



Computer Vision Design Example: 4K60 Dense Optical

	Xilinx ZU9
Frames/s	60
Power (W)	4.8
Latency (ms)	16.7
Utlization	15%

```
main(){
 imread(A);
 imread(B);
 denseOpticalFlowPyrltr(A,B,out)
 imshow(out);}
```

ZCU102 EV Platform

SDSoC Environment

SDSoC Generated

Platform

DMA

AXI-S

ZYNQ

Application

Libraries

Drivers

Linux

PS

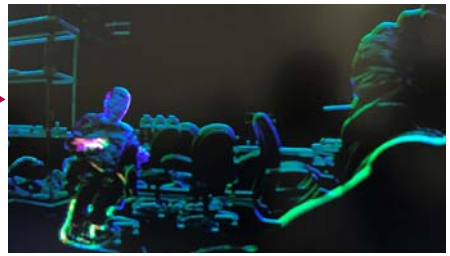
AXI

PL

MIPI

denseOpticalFlowPyrltr

HDMI



- nVidia number using CUDA OpenCV
- Both Xilinx and nVidia benchmarks do not include the camera inputs and HDMI/DP
- LK dense optical flow, non-pyramidal, non-iterative, Window size 53x53

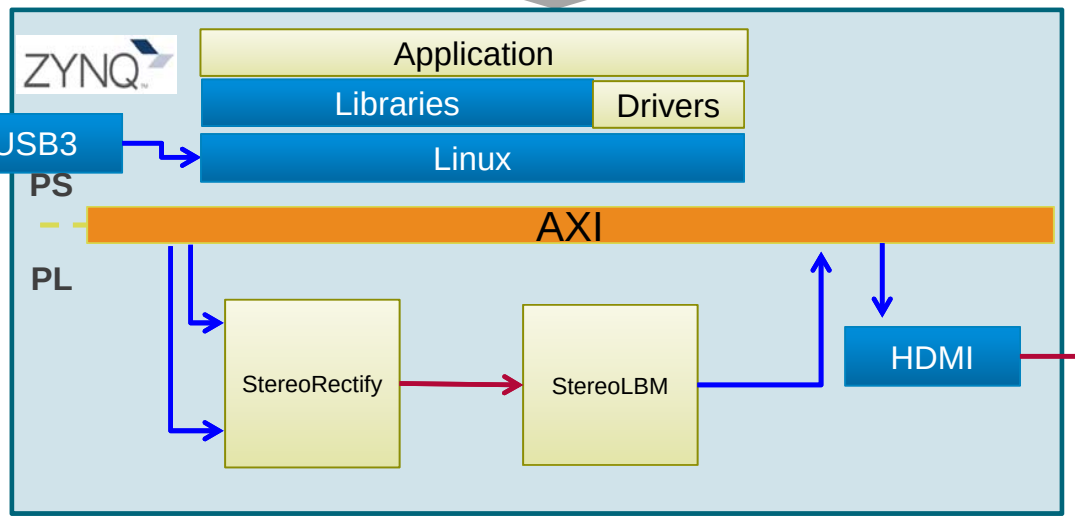
Computer Vision Design Example: Stereo Disparity Map

	Xilinx ZU9
Frames/s	140
Power (W)	4.8
Latency (ms)	7.1
Utilization	14%

```
main(){
 imread(A);
 imread(B);
 stereoRectify(A,B,C,D);
 stereoLBM(C,D,out);
 imshow(out);}
}
```

ZCU102 EV Platform

SDSoC Environment

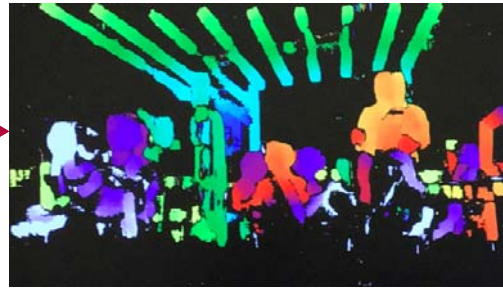


SDSoC Generated

Platform

DMA

AXI-S



- nVidia number using CUDA OpenCV
- SAD based stereo localBM
- Both Xilinx and nVidia benchmarks do not include the camera inputs and HDMI/DP outputs

What’s available now for OpenCV of Revision Stack

Design Examples on Xilinx.com/revision

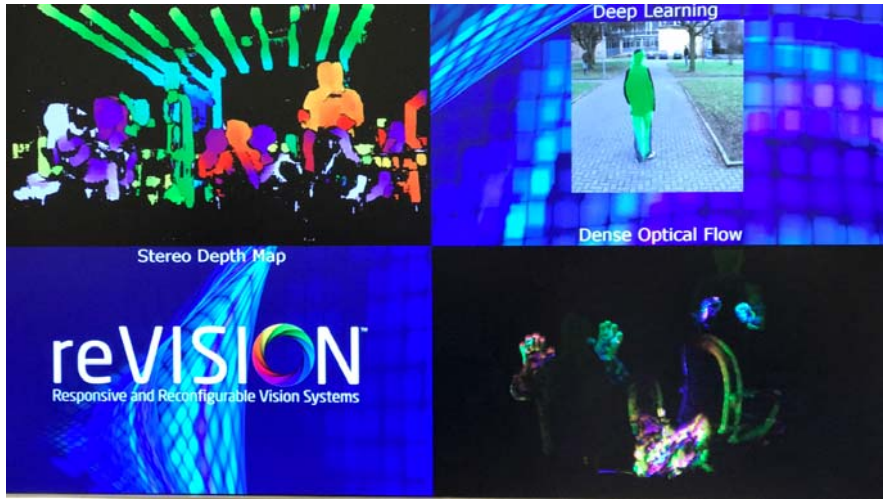
Xilinx Webpage ➔ Developer Zone ➔ Revision Zone

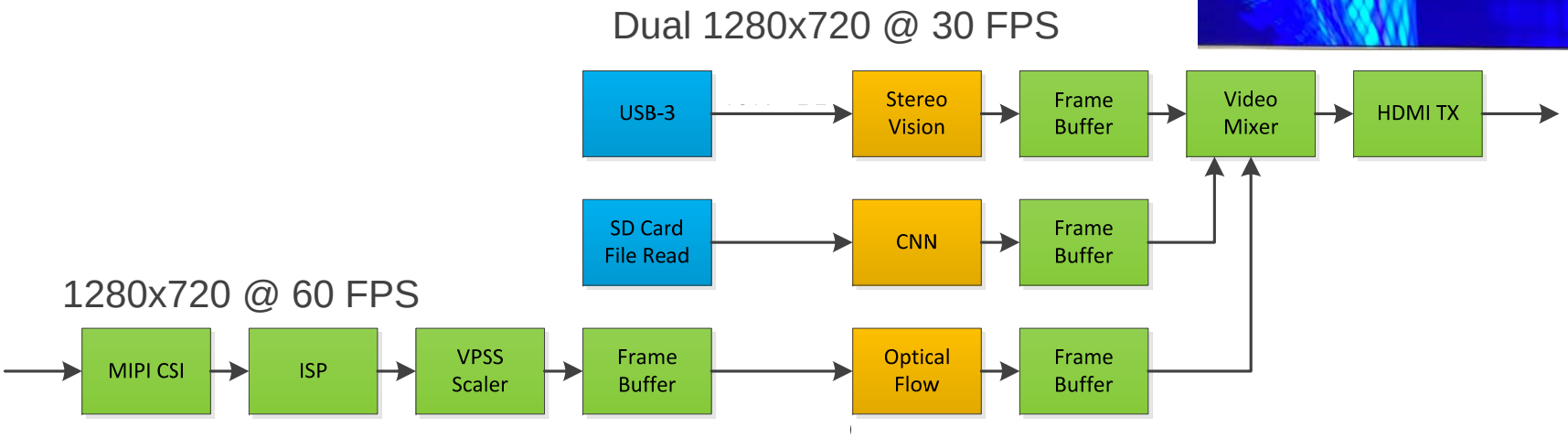
Computer Vision Design Examples

Design Example Provided by Xilinx	Latest SDSoC Version Supported	Board & SOM Supported	Provider
LK Dense Optical Flow iterative and pyramidal based implementation doing motion segmentation	2017.2	ZCU102 • Download (Production Silicon) • Download (ES2 Silicon) ZC702 • Download ZC706 • Download	Xilinx
Stereo Disparity Map Calculates disparity map from two sensor inputs using local block matching			
Warp Transform			
Harris Corner			
Bilateral Filter			

OpenCV and DNN live Demo with Revision

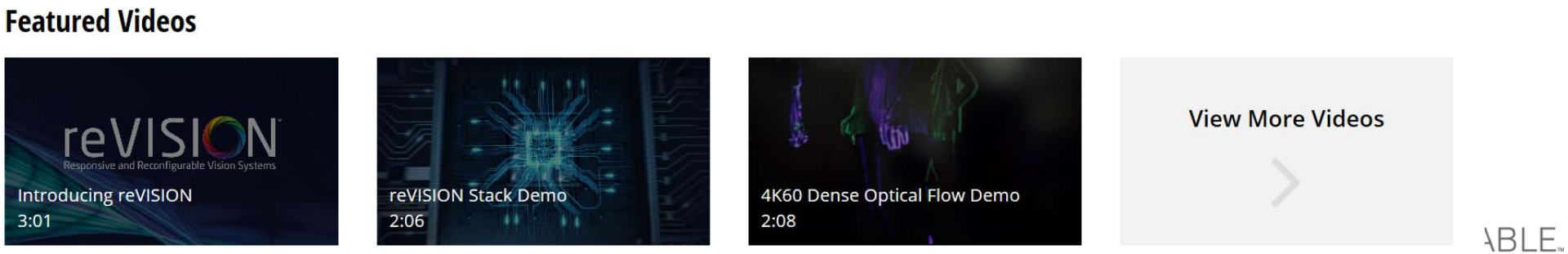
Putting It All Together: CV, CNN with Multiple Sensors





Summary

- Zynq SoCs offer superior performance and lower latency compared to other SoC offerings
- reVISION stack on SDSoC introduces familiar software environment with pre-optimized OpenCV libraries
- Available NOW
- Visit the reVISION developer zone https://www.xilinx.com/products/design-tools/embedded-vision-zone.html



Q&A