

reVISION™

Responsive and Reconfigurable Vision Systems



MACHINE LEARNING

|

COMPUTER VISION

|

SENSOR FUSION

CONNECTIVITY



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

Kamran Khan, Product Manager, Software Acceleration and Libraries
July 2017

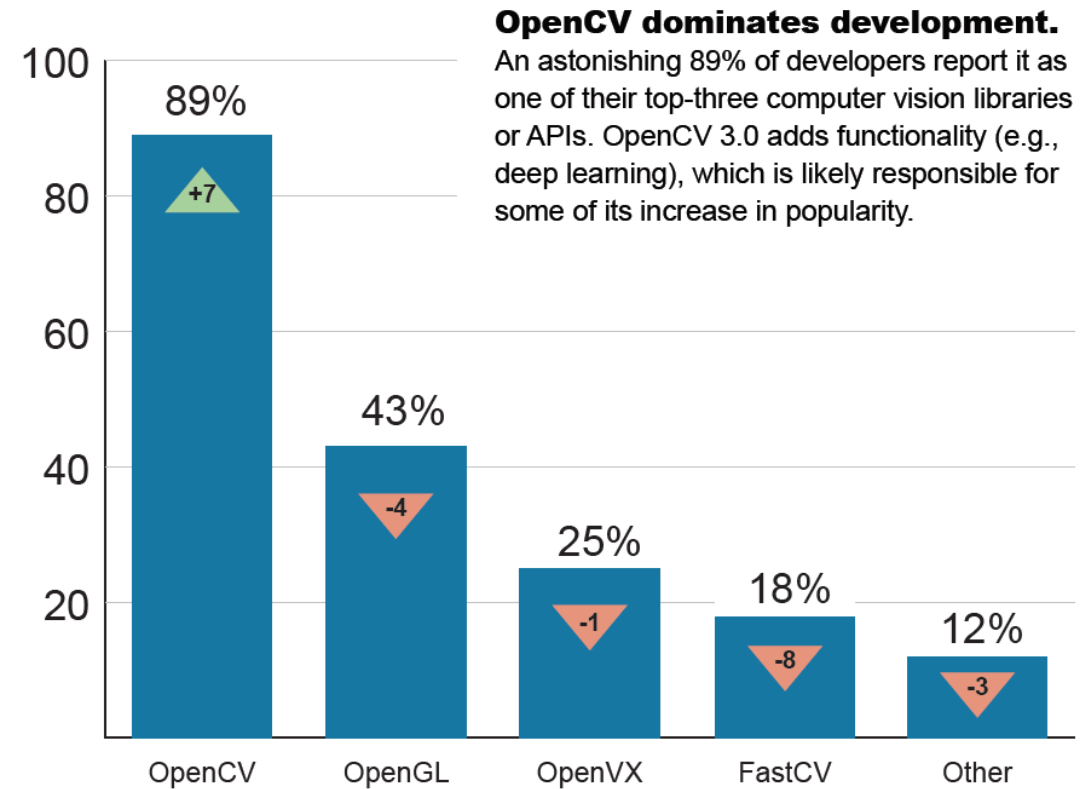
Agenda

- Why Zynq SoCs for Traditional Computer Vision
- Automated Flow for OpenCV HW Acceleration
- Case Study

OpenCV Needs Acceleration in Embedded

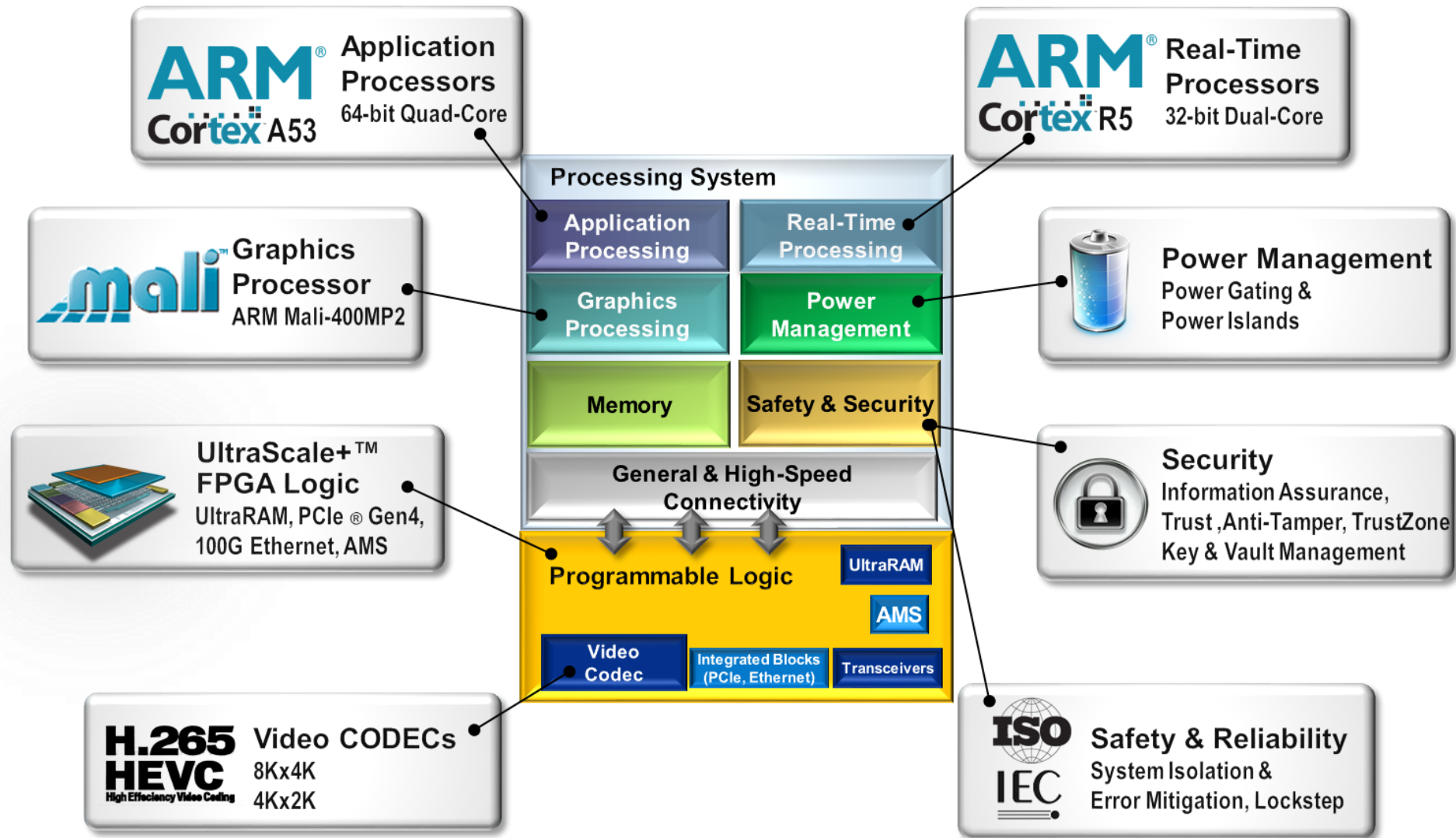
➤ Typical ARM Cortex-A53

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

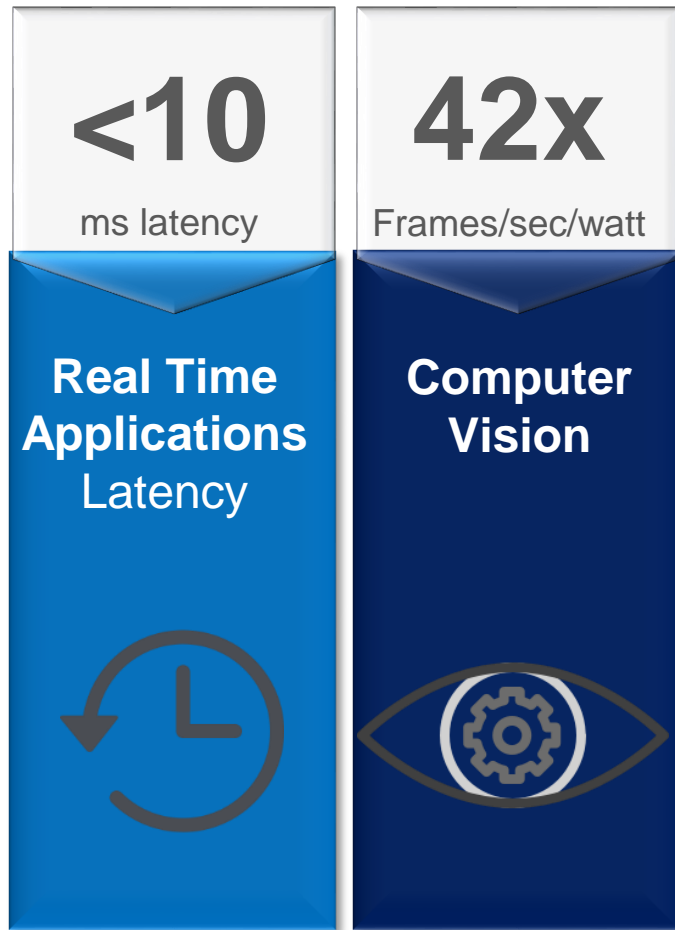


Source: Embedded Vision Alliance,
Embedded Vision Developer Survey, January 2017

Zynq Offers the Most Efficient CV Acceleration



Zynq Offer Superior Performance, Latency



Xilinx
Benchmark

Xilinx
Benchmark

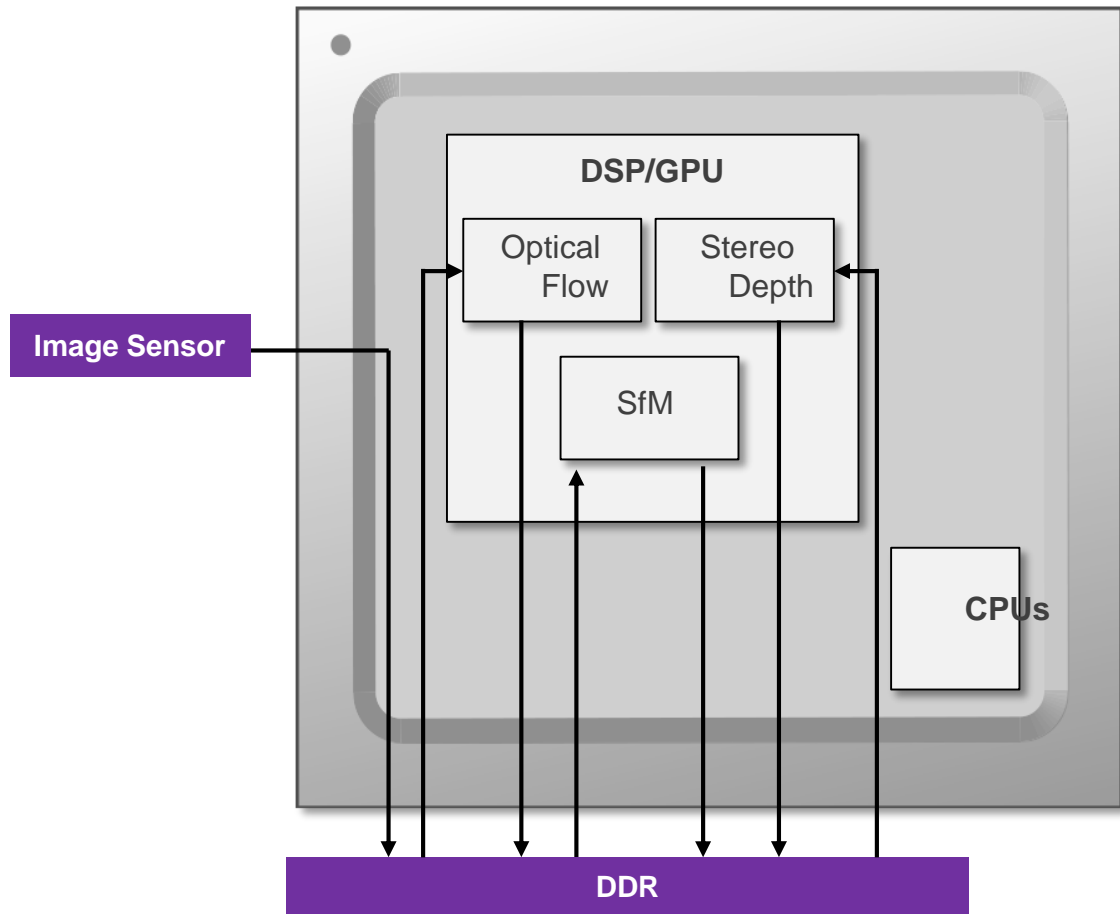
		Xilinx ZU9	Xilinx ZU5	eGPU*
CV: StereoLBM @1080p	Frames/s	700	296	43
	Power (W)	4.8	3.3	7.9
	Frames/s/watt	145.8	89.7	5.4

		Xilinx ZU9	Xilinx ZU5	eGPU*
CV: LK Dense Optical Flow @720p	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%)

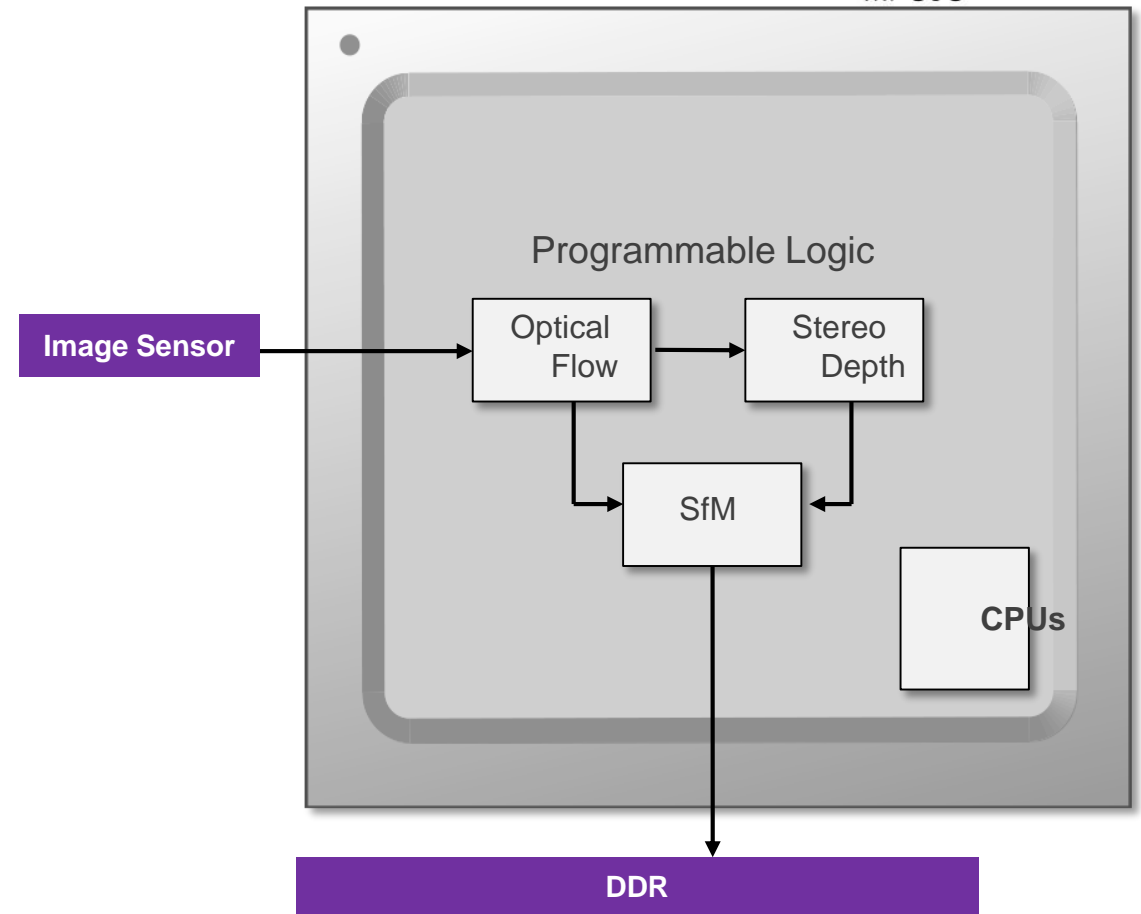
Why So Good? Efficient Window-based Streaming

Typical SoC



ZYNQ

ZYNQ
MPSoC



reVISION Stack



Caffe

Frameworks



SDSoC
Environment

DNN

CNN

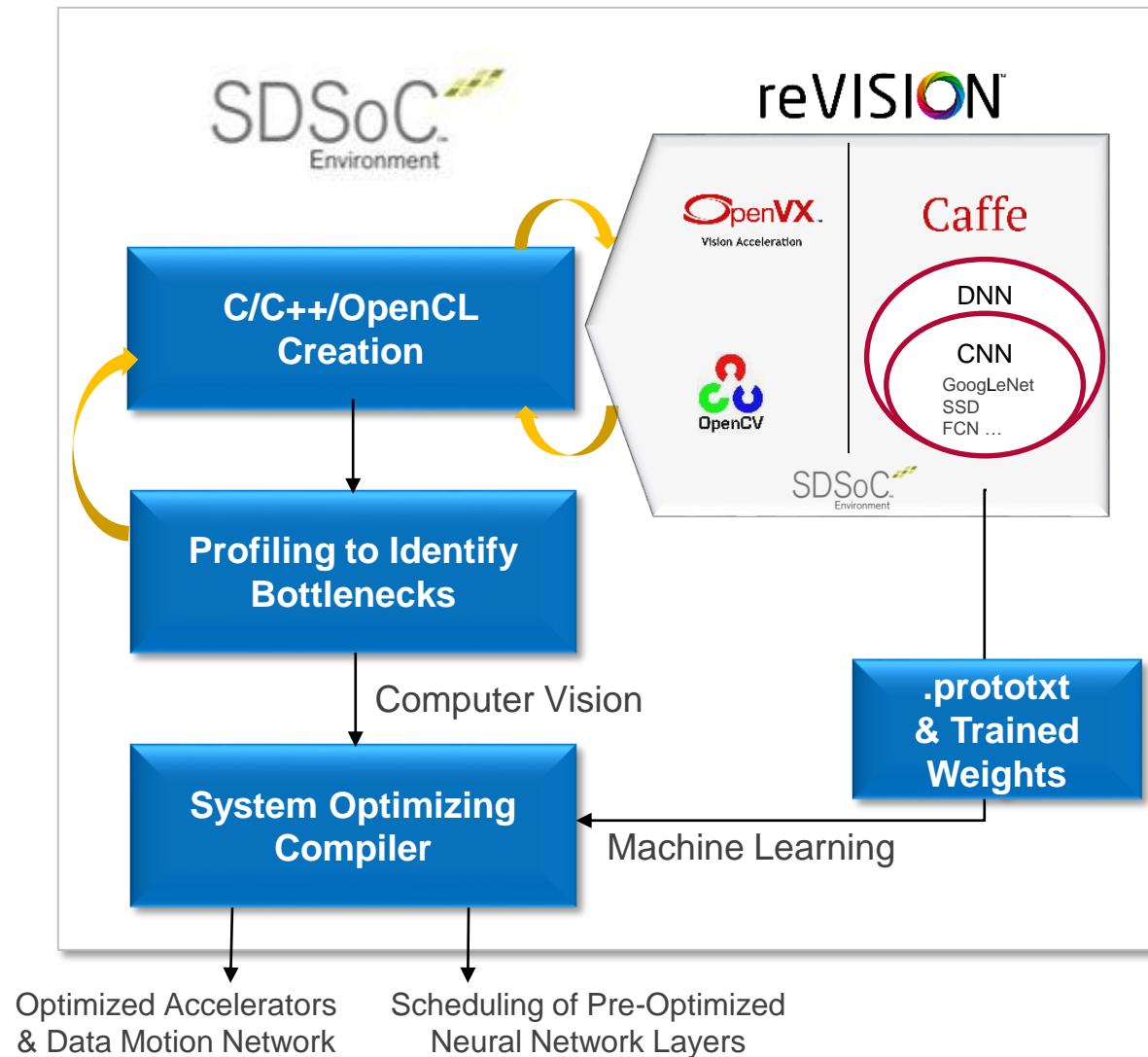
GoogLeNet
SSD
FCN ...

Libraries and Tools



Development Kits

Debunking “Zynq SoC is Hard to Program”



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

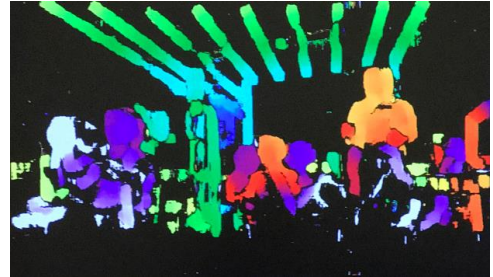


```
main() {
    cv::imread(A);
    cv::stereoRectify(A,B,C,D);
    cv::stereoLBM(C,D,out);
    cv::imshow(out);
}
```

Time [%]	Time
100.0	4504 ms
100.0	4500 ms
97.1	4370 ms
97.1	4370 ms
77.2	3474 ms
77.2	3473 ms
77.2	3471 ms
75.8	3412 ms
57.7	2597 ms
50.5	2274 ms
43.0	1933 ms
42.1	1895 ms
39.6	1782 ms
35.4	1593 ms
34.8	1567 ms
32.1	1444 ms

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);
}
```



xfOpenCV: HW Accelerated OpenCV Functions

Level 1		Level 2		Level 3
Absolute difference	Channel combine	Box	Scale/Resize	Histogram of Oriented Gradients (HOG)
Accumulate	Channel extract	Gaussian	StereoRectify	
Accumulate squared	Color convert	Median	Warp Affine	SVM (binary)
Accumulate weighted	Convert bit depth	Sobel	Warp Perspective	OTSU Thresholding
Arithmetic addition	Table lookup	Custom convolution	Fast corner	Mean Shift Tracking (MST)
Arithmetic subtraction	Histogram			LK Dense Optical Flow
Bitwise: AND, OR, XOR, NOT	Gradient Phase	Dilate	Harris corner	Canny edge detection
Pixel-wise multiplication	Min/Max Location	Erode	Remap	Image pyramid
Integral image	Mean & Standard Deviation	Bilateral	Equalize Histogram	Color Detection
Gradient Magnitude	Thresholding			StereoLBM

Custom CV Function / Library Creation Flow

1

Cross-compile to Zynq (ARM A9/A53)

2

Write custom CV function in C, C++ or OpenCL.
Optimize for hardware using HLS

3

Assign functions to hardware.
Compile using SDSoc

4

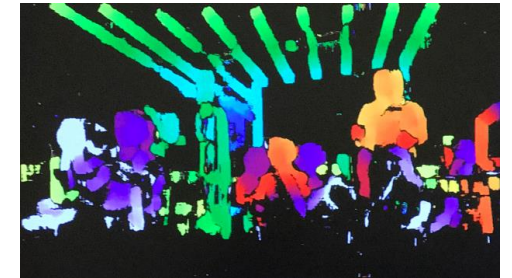
Run on a Zynq board

```
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

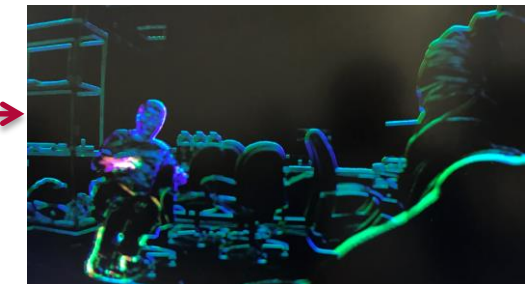
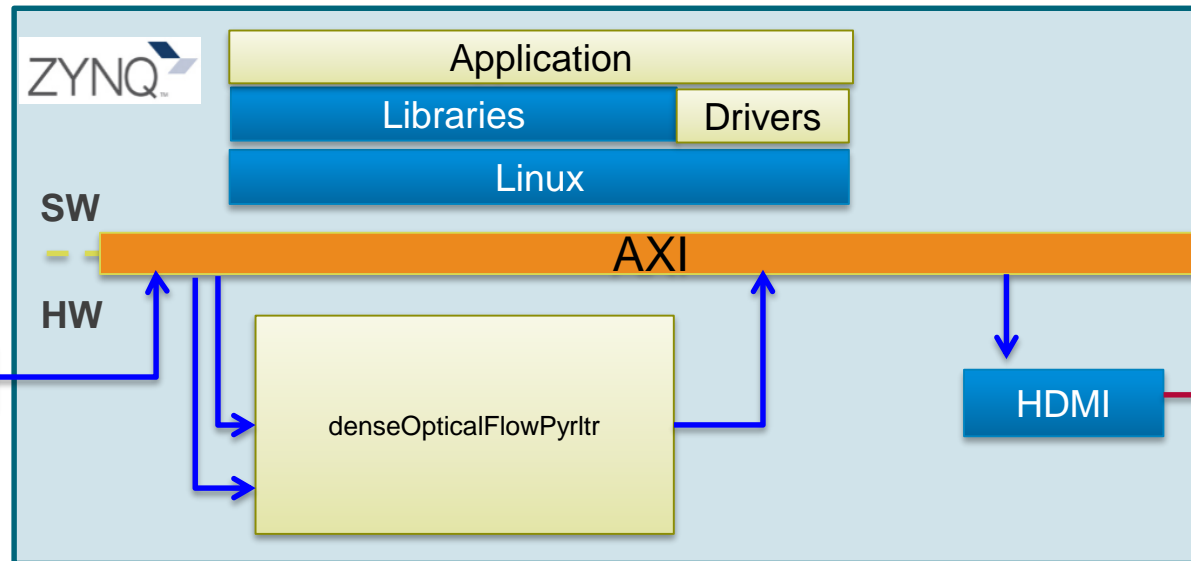


Example: 4K60 LK Dense Optical Flow

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

```

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



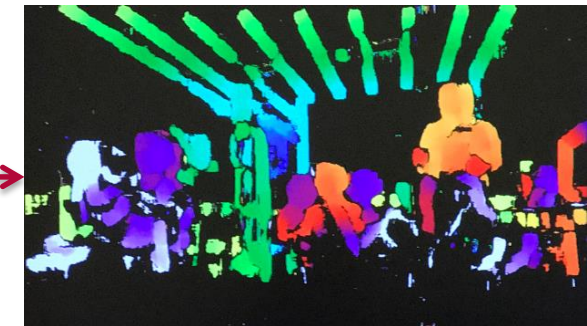
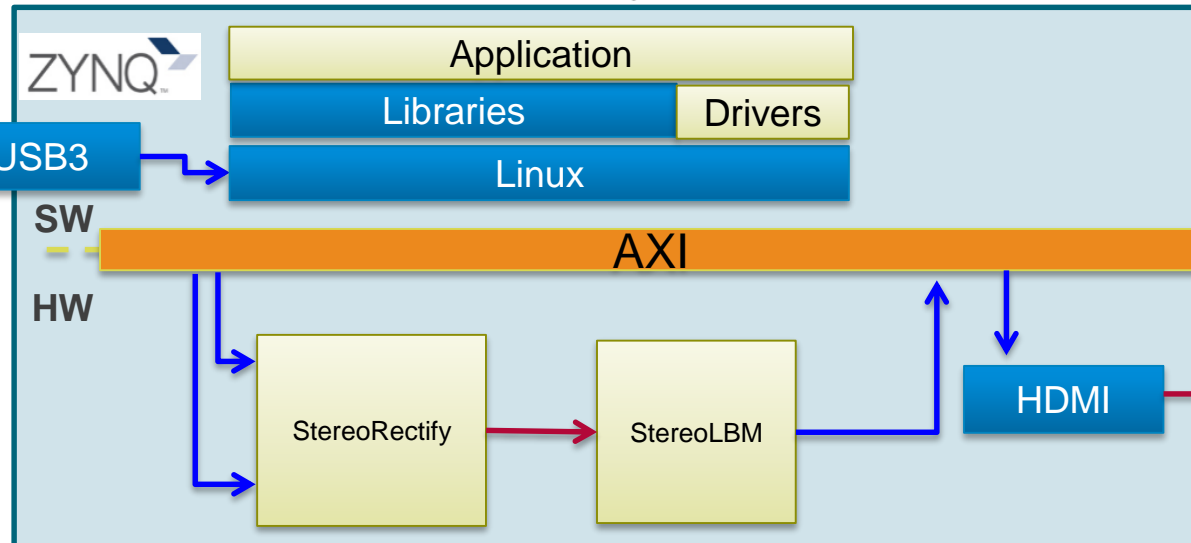
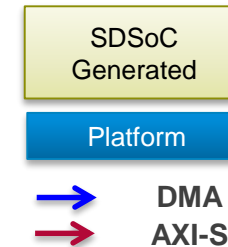
- nVidia number using CUDA OpenCV
- Both Xilinx and nVidia benchmarks do not include the camera inputs and HDMI/DP
- LK dense optical flow, non-pyramidal, non-iterative, Window size 53x53

Example: Stereo Depth Map

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

```

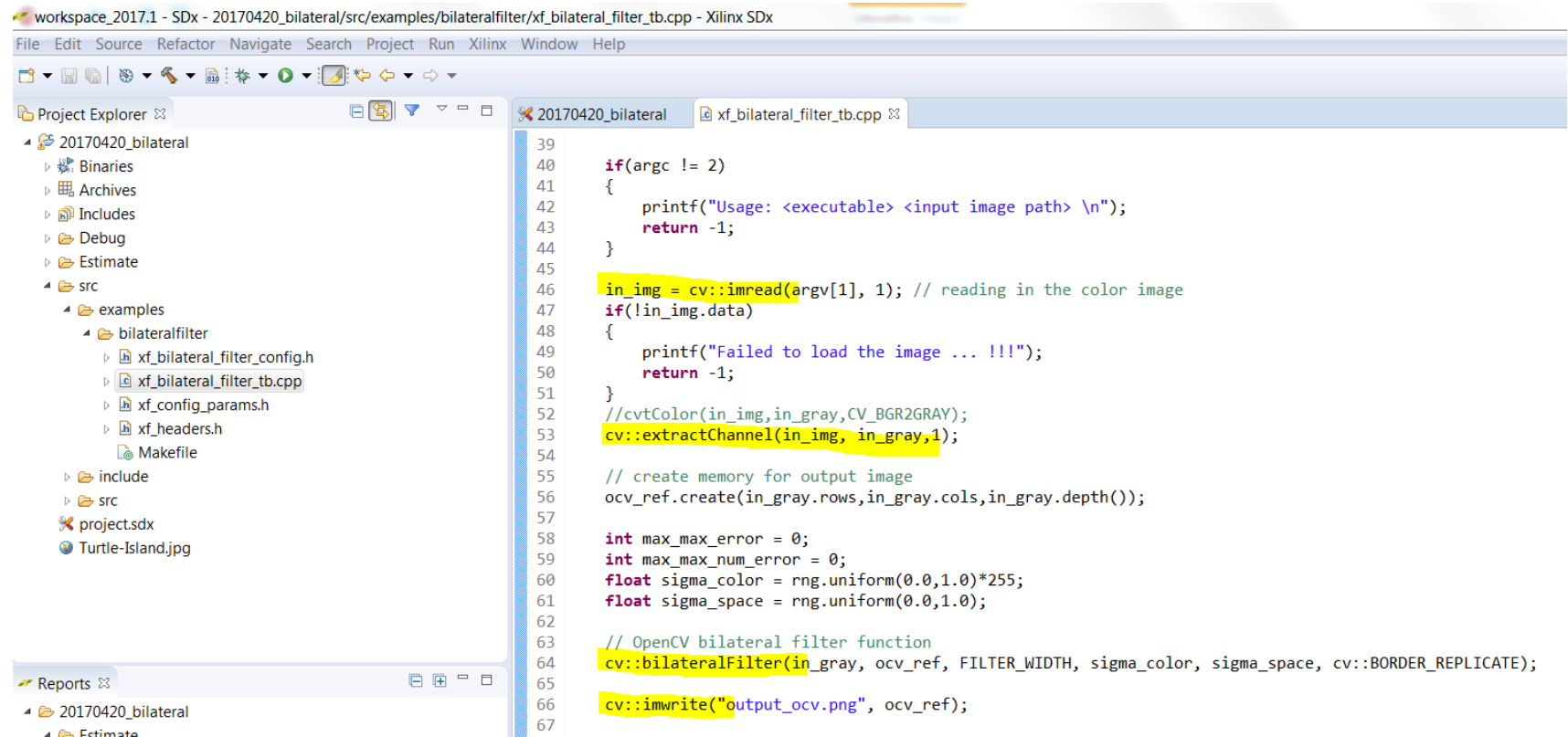
main() {
    imread(A);
    imread(B);
    stereoRectify(A,B,C,D);
    stereoLBM(C,D,out);
    imshow(out);
}
    
```



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

Step 1: Port Desktop OpenCV Application to Zynq

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



The screenshot displays the Xilinx IDE interface. The Project Explorer on the left shows a project named '20170420_bilateral' with a sub-directory 'src' containing an 'examples' folder. The 'examples' folder contains a sub-directory 'bilateralfilter' with files: 'xf_bilateral_filter_config.h', 'xf_bilateral_filter_tb.cpp', 'xf_config_params.h', and 'xf_headers.h'. The main editor window shows the source code for 'xf_bilateral_filter_tb.cpp'. The code includes standard C++ headers and OpenCV headers. It defines a function that takes an input image path as an argument. The code reads the image in grayscale, checks if it was loaded successfully, and then applies a bilateral filter. The filtered image is then written to a file named 'output_ocv.png'. The code is as follows:

```
39
40  if(argc != 2)
41  {
42      printf("Usage: <executable> <input image path> \n");
43      return -1;
44  }
45
46  in_img = cv::imread(argv[1], 1); // reading in the color image
47  if(!in_img.data)
48  {
49      printf("Failed to load the image ... !!!");
50      return -1;
51  }
52  //cvtColor(in_img,in_gray,CV_BGR2GRAY);
53  cv::extractChannel(in_img, in_gray,1);
54
55  // create memory for output image
56  ocv_ref.create(in_gray.rows,in_gray.cols,in_gray.depth());
57
58  int max_max_error = 0;
59  int max_max_num_error = 0;
60  float sigma_color = rng.uniform(0.0,1.0)*255;
61  float sigma_space = rng.uniform(0.0,1.0);
62
63  // OpenCV bilateral filter function
64  cv::bilateralFilter(in_gray, ocv_ref, FILTER_WIDTH, sigma_color, sigma_space, cv::BORDER_REPLICATE);
65
66  cv::imwrite("output_ocv.png", ocv_ref);
67
```

Step 2: Assign Functions to Hardware Acceleration

- Minor mods needed to use OpenCV libraries for hardware acceleration
 - Namespace change: “cv::” to “xF::”
 - Add template parameters for optimized hardware generation
- Simply assign critical functions to hardware

```
uint16_t width = in_gray.cols;
uint16_t height = in_gray.rows;

xF::Mat<XF_8UC1, HEIGHT, WIDTH, NPC1> _src(height,width);
xF::Mat<XF_8UC1, HEIGHT, WIDTH, NPC1> _dst(height,width);

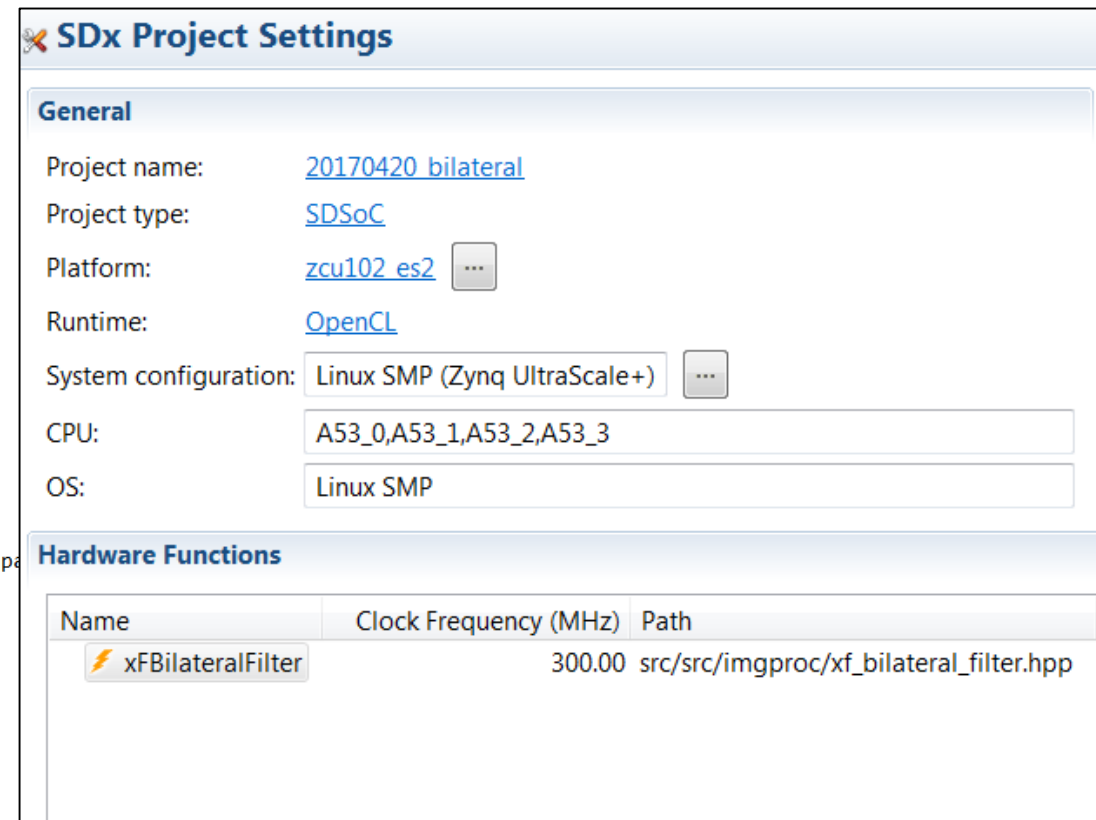
_src.copyTo(in_gray.data);

xF::BilateralFilter<XF_FILTER_WIDTH, XF_BORDER_REPLICATE, XF_8UC1, HEIGHT, WIDTH, NPC1>(_src,_dst, sigma_spa

out_img.data = _dst.copyFrom();
imwrite("output_hls.png", out_img);

absdiff(ocv_ref, out_img, diff); // Compute absolute difference image

// Find minimum and maximum differences.
```



The screenshot shows the 'SDx Project Settings' dialog box. The 'General' tab is active, displaying the following configuration:

- Project name: 20170420 bilateral
- Project type: SDSoc
- Platform: zcu102_es2
- Runtime: OpenCL
- System configuration: Linux SMP (Zynq UltraScale+)
- CPU: A53_0,A53_1,A53_2,A53_3
- OS: Linux SMP

The 'Hardware Functions' tab is also visible, showing a table with the following entries:

Name	Clock Frequency (MHz)	Path
xBilateralFilter	300.00	src/src/imgproc/xf_bilateral_filter.hpp

Step 3: 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

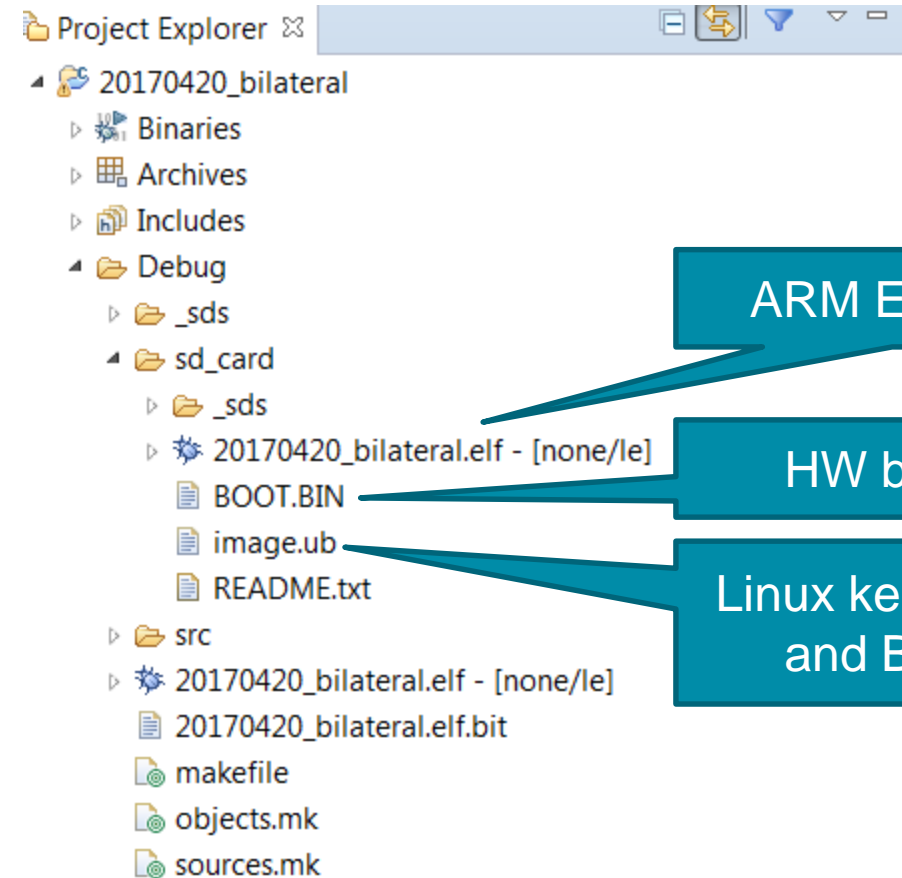
Details

Performance estimates for 'xFBilateralFilter_3_1_0_1080_1 ...'

Hardware accelerated (Estimated cy) 30185245

Resource utilization estimates for Hardware functions

Resource	Used	Total	% Utilization
DSP	22	2520	0.87
BRAM	3	912	0.33
LUT	7061	274080	2.58
FF	6627	548160	1.21

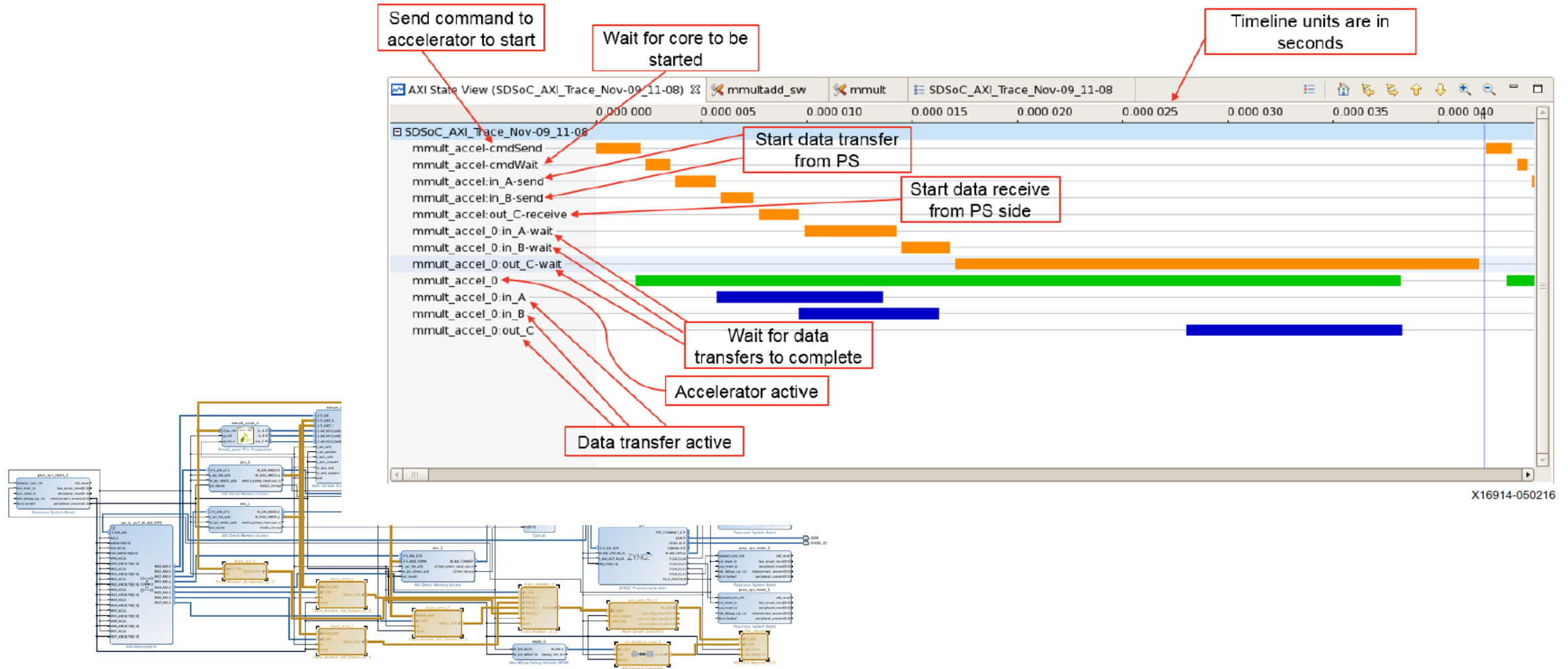


ARM Executable

HW bitstream

Linux kernel, Rootfs
and Boot files

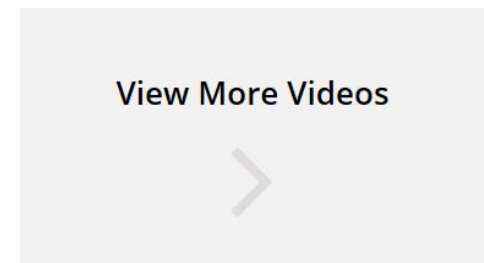
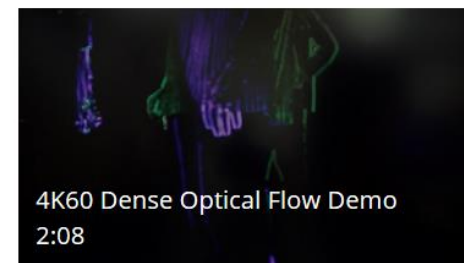
Step 4: Run on a Board and Collect Traces



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#computer>

Featured Videos



Design Examples on Xilinx.com/revision

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	2017.1	ZCU102, ZC702, ZC706	Xilinx
Stereo Disparity Map Calculates disparity map from two sensor inputs using local block matching	2017.1	ZCU102, ZC702, ZC706	Xilinx
Warp Transform	2017.1	ZCU102, ZC702, ZC706	Xilinx
Harris Corner	2017.1	ZCU102, ZC702, ZC706	Xilinx
Bilateral Filter	2017.1	ZCU102, ZC702, ZC706	Xilinx

Resources

- [INT8 Whitepaper](#)
- [Machine Learning Whitepaper](#)
- [reVISION Backgrounder](#)
- [Additional Papers & Tutorials](#)
- [Xilinx Embedded Vision Videos](#)
- [Forums](#)

For all this and more, visit [Xilinx.com/reVISION](https://www.xilinx.com/reVISION)

Empowering Product Creators to Harness Embedded Vision



The Embedded Vision Alliance (www.Embedded-Vision.com) is a partnership of 60+ leading embedded vision technology and services suppliers

Mission: Inspire and empower product creators to incorporate visual intelligence into their products

The Alliance provides low-cost, high-quality technical educational resources for product developers

Register for updates at www.Embedded-Vision.com

The Alliance enables vision technology providers to grow their businesses through leads, ecosystem partnerships, and insights

For membership, email us: membership@Embedded-Vision.com



Embedded Vision Insights
The Latest Developments on Designing Machines that See

Learn How to Develop Deep Learning Applications for Computer Vision in TensorFlow

Full-Day, Hands-On Training Class



Deep Learning for Computer Vision with TensorFlow

Topics:

- Introduction to TensorFlow
- TensorBoard Visualization Tools
- Open Source CNN Models
- Neural Networks in TensorFlow
- Object Recognition in TensorFlow
- Using TensorFlow in Embedded Systems



July 13, 2017 • Hyatt Regency Santa Clara • Santa Clara, California
September 7, 2017 • Steigenberger Hotel • Hamburg, Germany

<http://bit.ly/2pDRjk4>

Q&A