XDF XILINX DEVELOPER FORUM

SDSoC, ML, and Embedded Vision

Presented By

Rob Armstrong
Sr. Product Marketing Manager, Xilinx, Inc.
October 1-2, 2018

XILINX

Introduction

> The world is changing, adapting quickly to leverage machine learning
> Algorithms must scale from the cloud to the edge
> How can we quickly and easily develop and deploy complex systems?

reVISION Stack

Frameworks & Libraries

OpenCV

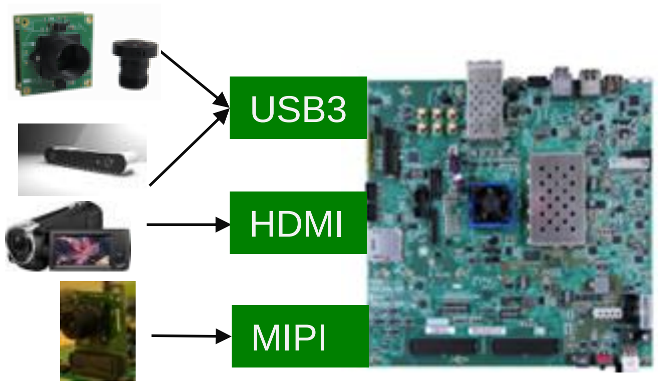
gstreamer

Machine Learning

Development tools

SDSoC Environment

Platforms



Xilinx Announces the Acquisition of DeePhi Tech

Deal to Accelerate Data Center and Intelligent Edge Applications

BEIJING and SAN JOSE, Calif., July 17, 2018 – Xilinx, Inc. (NASDAQ: XLNX), the leader in adaptive and intelligent computing, announced today that it has acquired DeePhi Tech, a Beijing-based privately held start-up with industry-leading capabilities in machine learning, specializing in deep compression, pruning, and system-level optimization for neural networks.

DEEPHi
深鉴科技

Integrated Xilinx-Deephi Roadmap

Xilinx AI Development

	Edge/Embedded			Cloud/DC
Models	20+ pruned / customized / basic models			
Software Stack	Deephi Pruning			
	Deephi Quantizer	SDSoC	SDAccel	Deephi Quantizer
	Deephi Compiler			xfDNN Compiler
	Deephi Runtime			xfDNN Runtime
FPGA IP	Deephi DPU	Deephi LSTM		xDNN

Platforms


Z7020 Board


Z7020 SOM


ZU2/3 SOM


ZU2/3 Card

ZU9 Card


ZCU102


ZCU104


Ultra96

Xilinx U200, U250, U280

awsmarketplace NIMBIX Alibaba Cloud aliyun.com HUAWEI

Development Environment

What is SDSoC

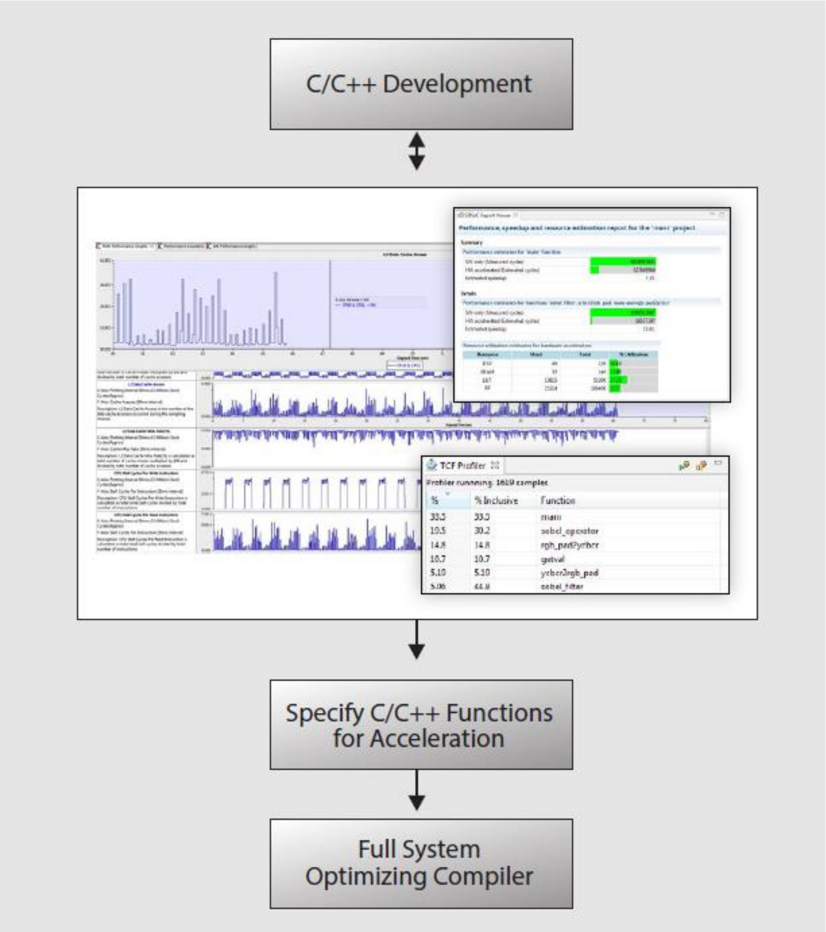
- **SDSoC – Software Defined System-on-Chip – is a development environment tailored to tightly coupled hardware/software designs**
- **Allows seamless integration of hardware and software**
- **Automates and streamlines memory allocation, cache management, DMA, and device interaction**

- **The SDx development environment provides the SDSoC and SDAccel tools with a common infrastructure**
  - Eclipse-based IDE with support for project creation, emulation, performance estimation, implementation, and debug
  - Implement heterogeneous embedded systems easily, combining multiple hardware accelerators, multiple applications, etc. all in one environment

Software/Hardware Interaction

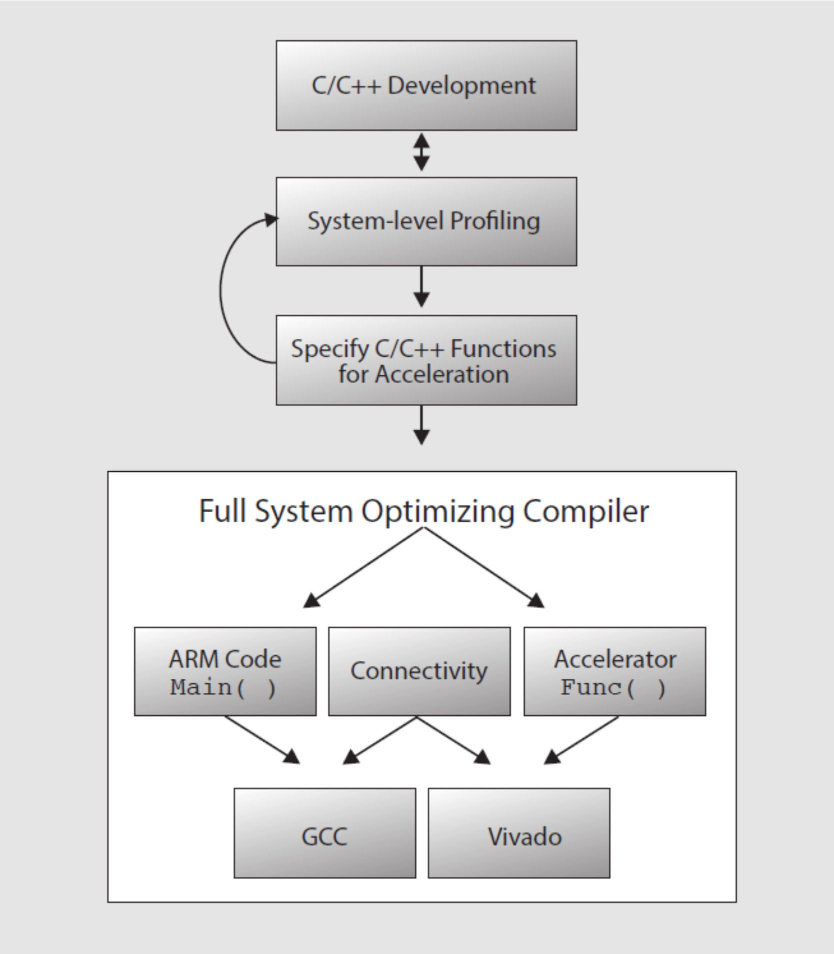
- **SDSoC can combine hardware from multiple sources**
  - Your own RTL, either in the platform or in the form of C-callable IP libraries
  - IP from third parties
  - IP generated with other Xilinx tools such as System Generator
  - The Xilinx high-level synthesis tool
- **Abstracting the mechanism of moving data between PS and PL (and using optimized methods) enables faster design**
- **In-depth system profiling and analysis tools help to identify and resolve system performance bottlenecks early and fix quickly**
- **Graph-based system analysis optimizes block-level connectivity**
  - Data is where it needs to be, when it needs to be there

SDSoC's System Level Profiling



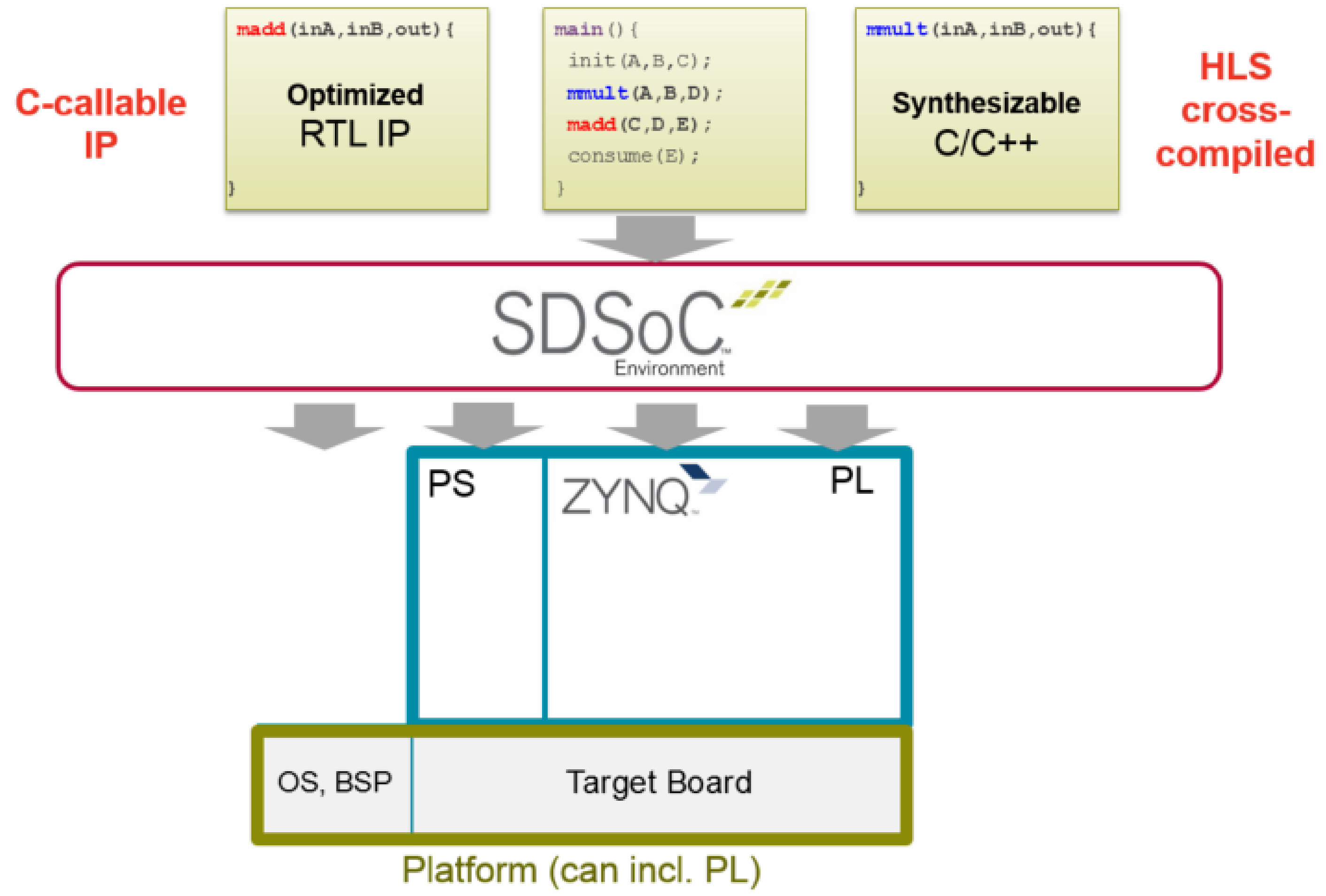
- > **Rapid system estimation in minutes**
  - No synthesis and place-n-route
  - Reports both performance and hardware utilization
- > **Automated performance measurement**
  - Runtime measurement of cache, memory, and bus utilization
  - HW-SW event trace showing time for accelerator IPs, data mover IPs and software overhead

Full System Optimizing Compiler

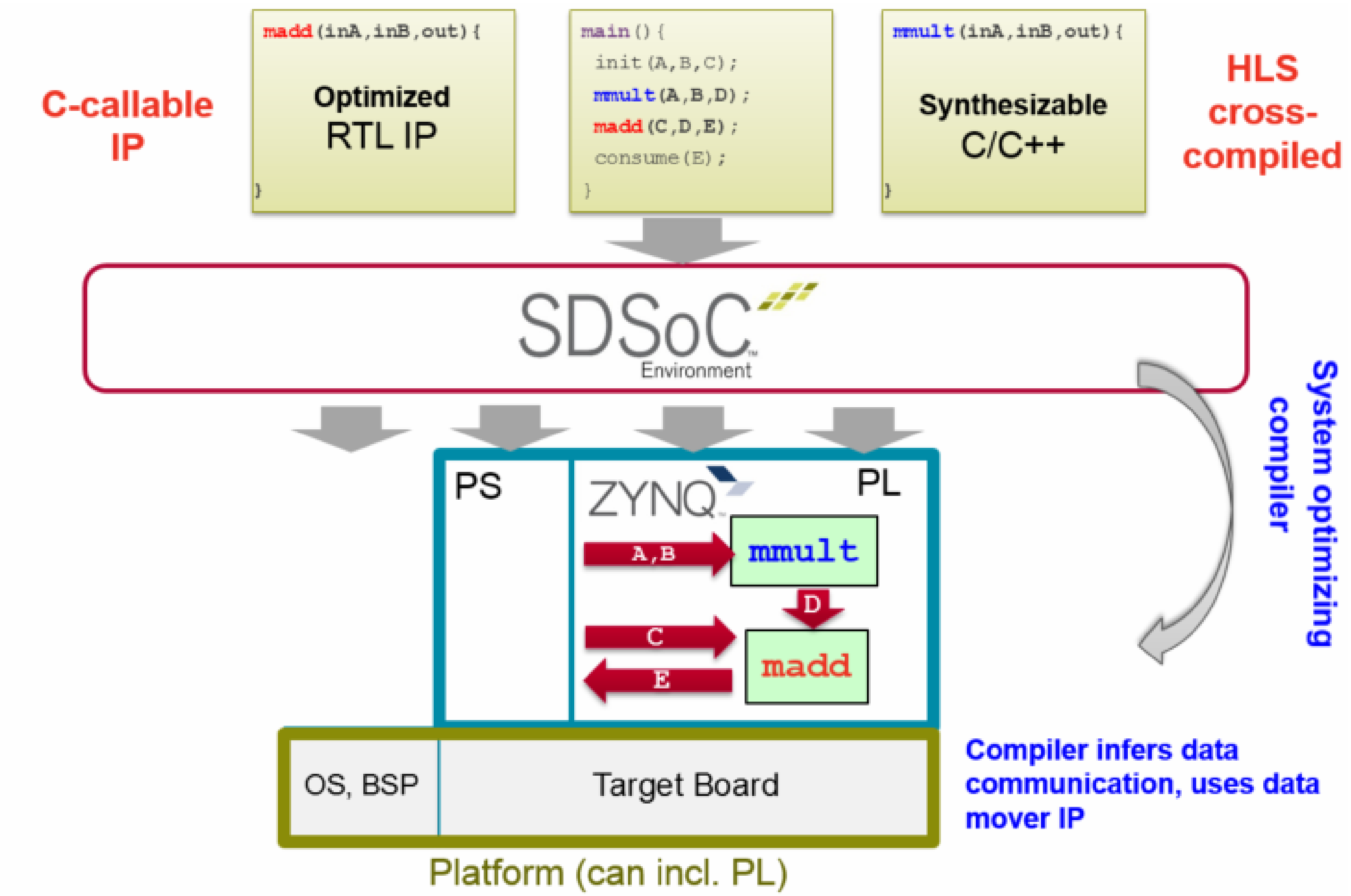


- **Full system from C/C++**
  - Automated function acceleration in PL
  - Up to 100X increase in performance vs. software
  - System optimized for latency, bandwidth, and hardware utilization
  - C/C++ pragma to override the connectivity architecture for power users

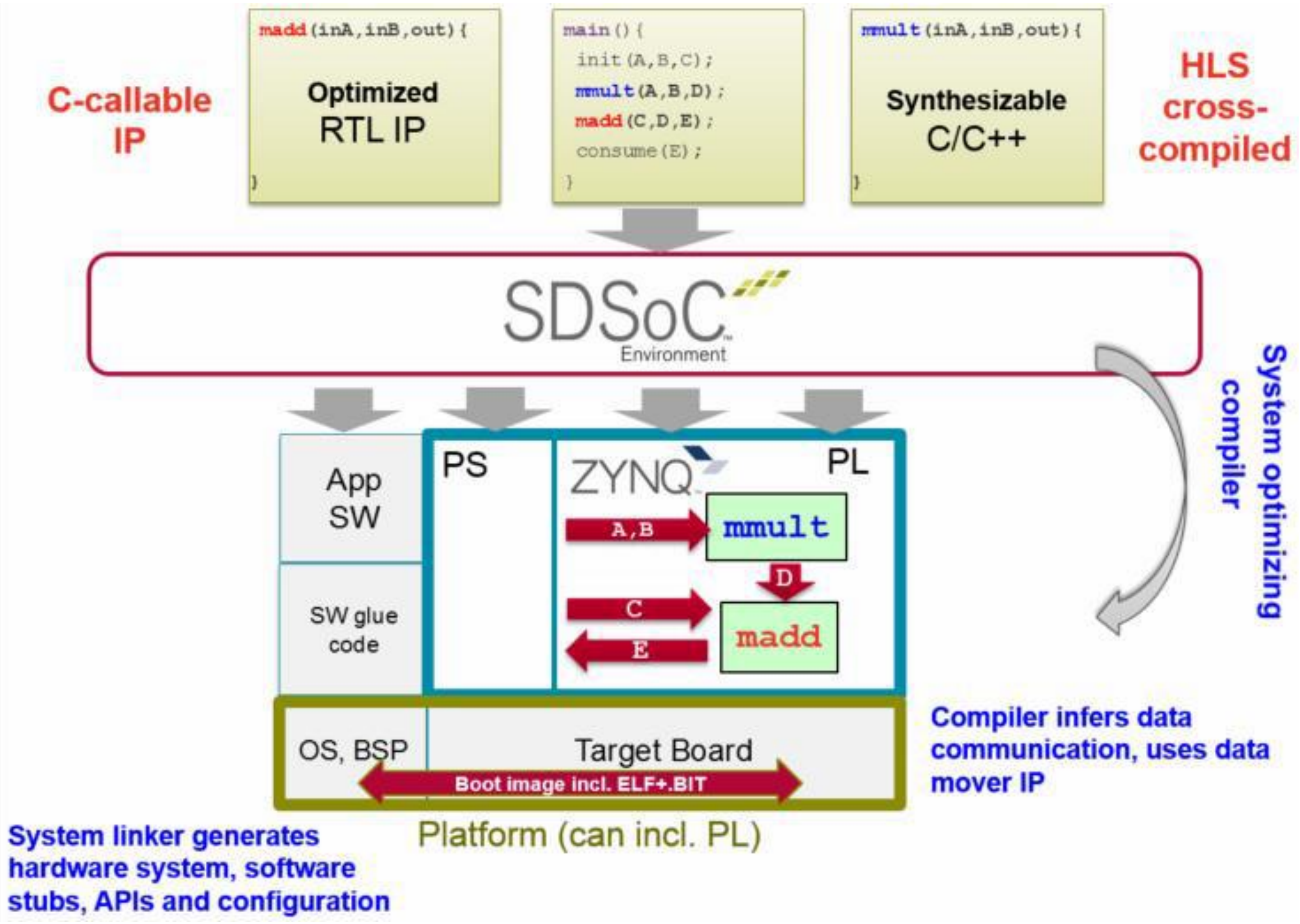
SDSoC Tool Flow



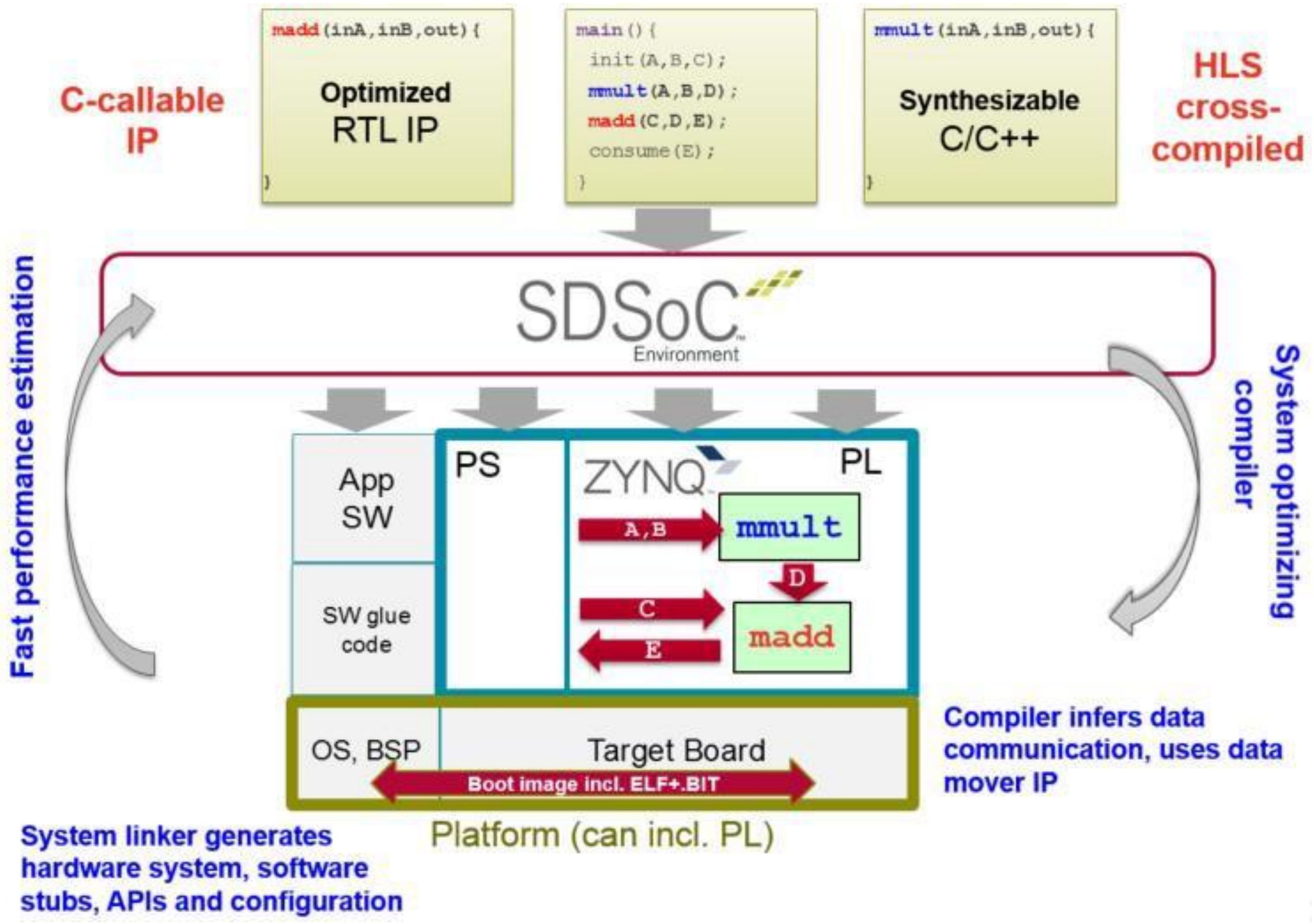
SDSoC Tool Flow



SDSoC Tool Flow



SDSoC Tool Flow



Designing with High-Level Synthesis

> Designing, verifying, and implementing IP with C-based design has never been easier

> Data flow analysis, timing analysis, and other tools make fine-tuning accelerators for performance easy

> Quick turn times and C-based unit verification help to catch functional errors early in the design cycle

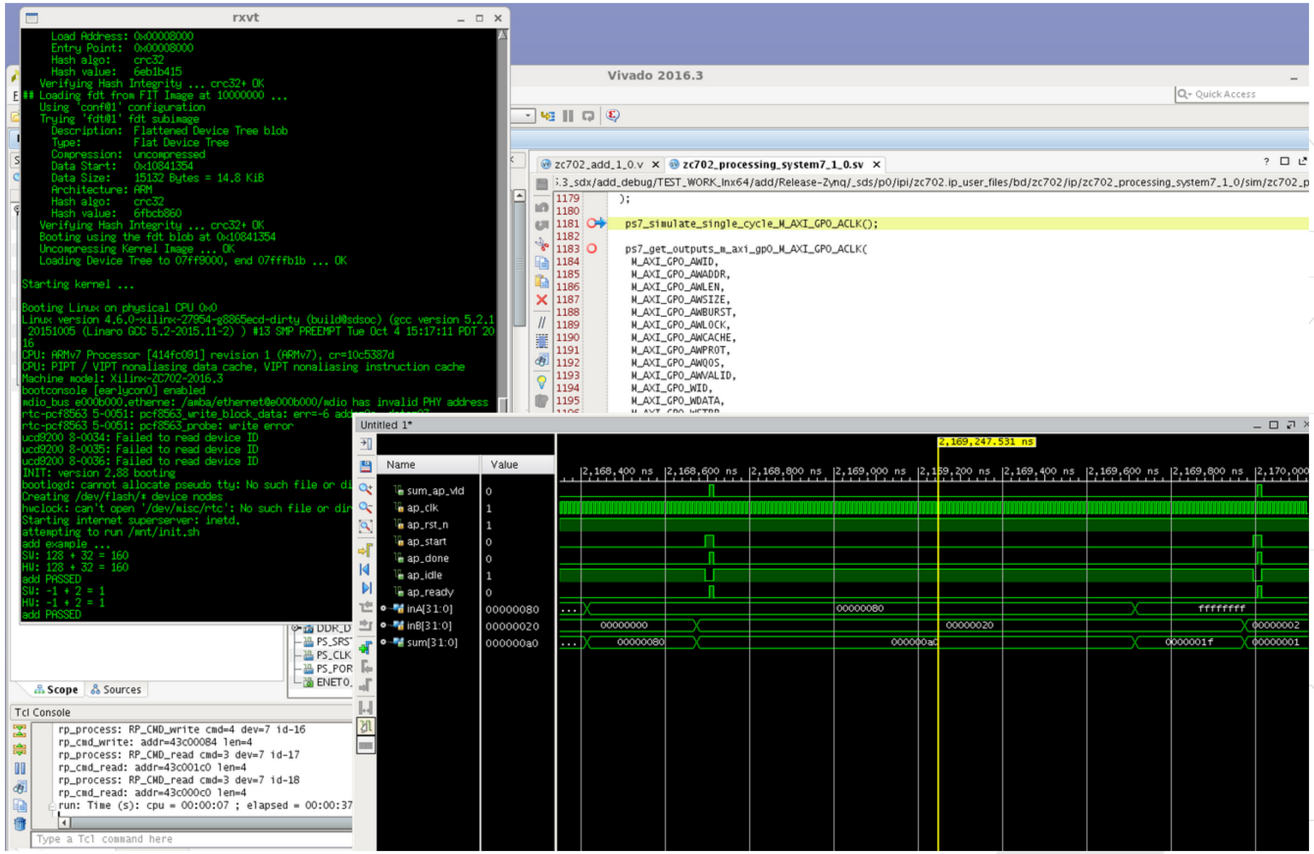
> Complete HW design environment using standard C/C++

QEMU / RTL emulation

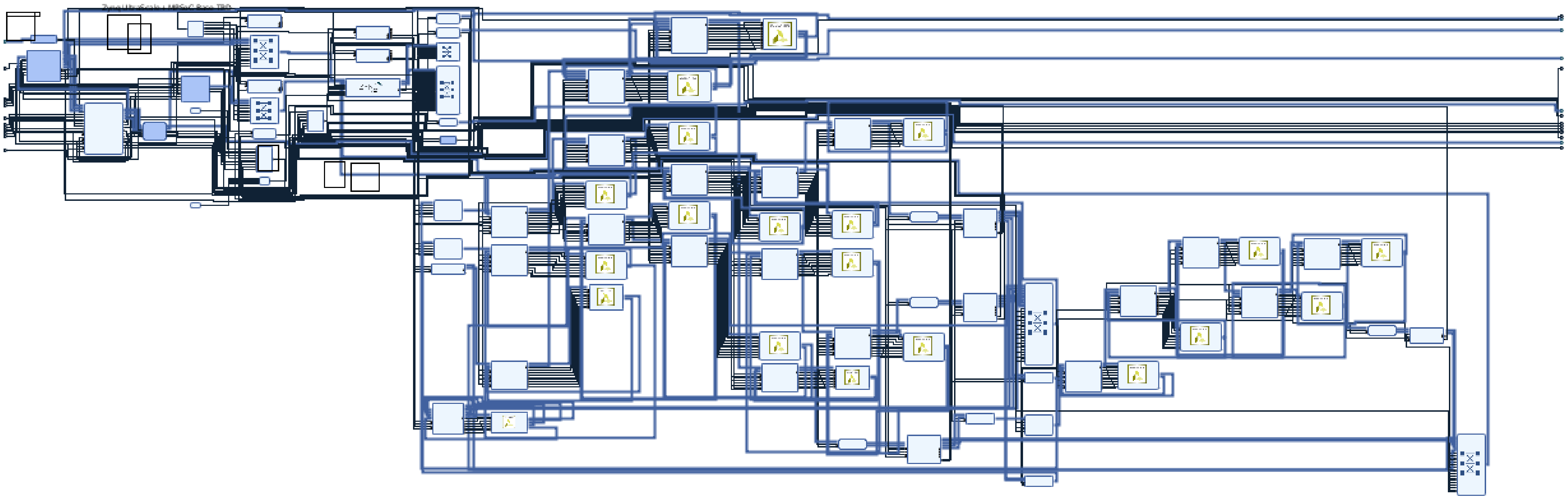
> Run software with generated RTL without firmware build

> User flow parallels hardware flow for build, debug, and launch run

> Visibility into hardware design after OS boot



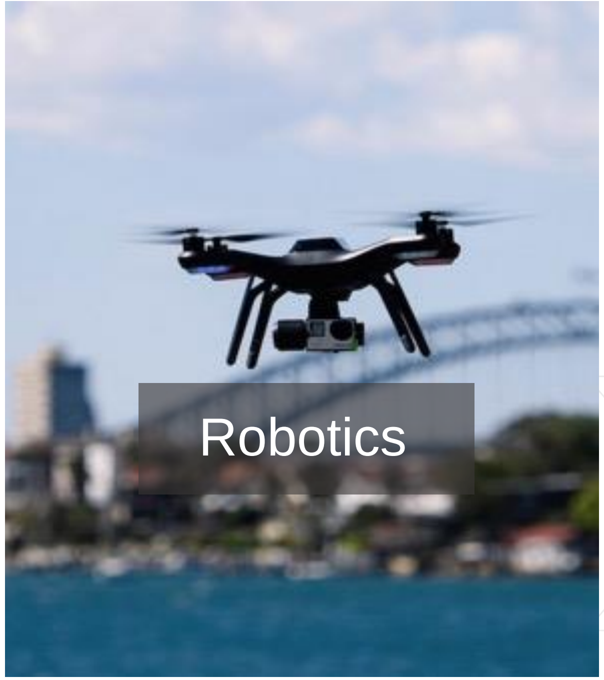
Can it Handle My Design?



Yes

Machine Learning

Machine Learning Applications for Xilinx

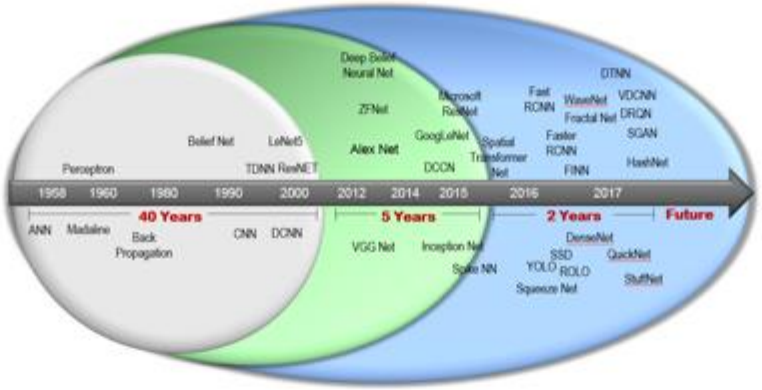


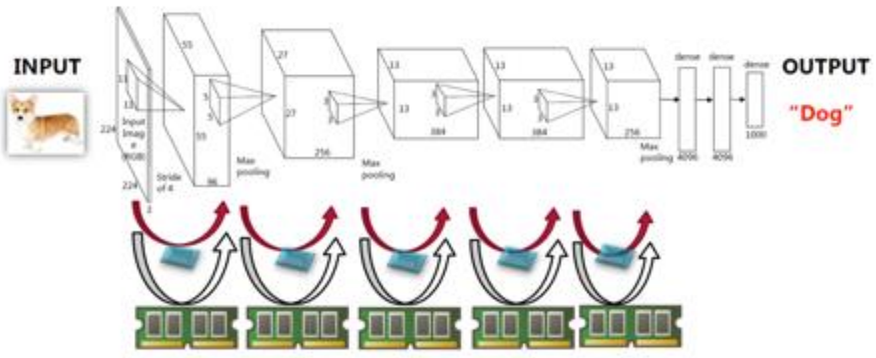
And there are many more …

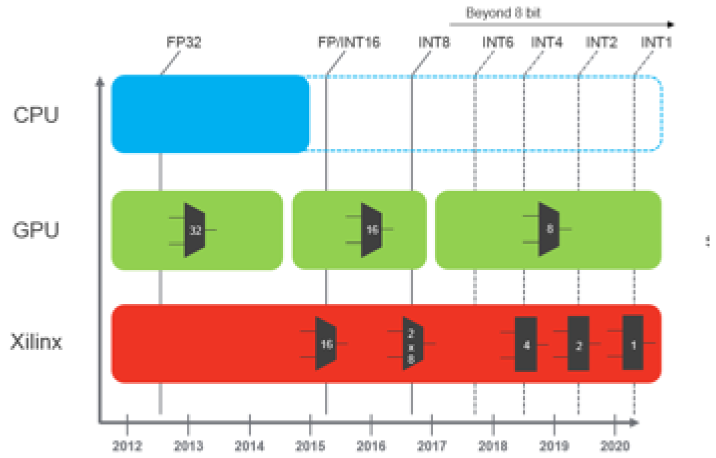
Edge ML

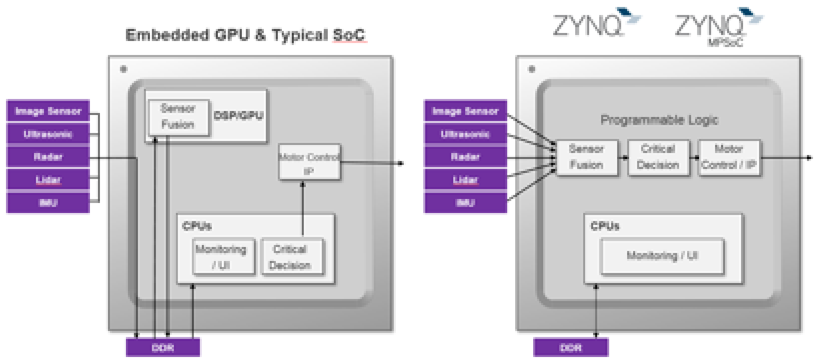
Recap of Xilinx Value Proposition in Edge ML

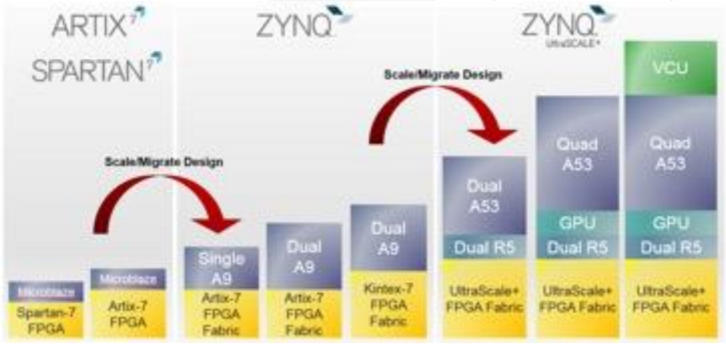
Xilinx offers the optimal tradeoff among latency, power, cost, flexibility, scalability & time-to-market for Edge ML



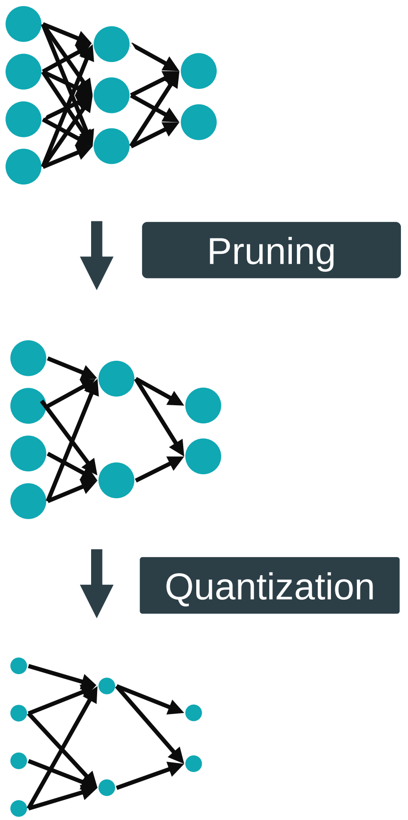








Unique, Patented Deep Learning Acceleration Techniques



> **Best paper awards for breakthrough DL acceleration**

> **Xilinx compression technology can:**

- >> Reduce DL accelerator footprint into smaller devices
- >> Increase performance per watt (higher performance and/or lower energy)

Unique Pruning Technology Provides a Significant Advantage

DeePhi Solution Stack for Deep Learning

Models

Face detection | Pose estimation | Video analytics | Lane detection | Object detection | Segmentation

Framework

Caffe | Darknet | TensorFlow

Tools & IP

DEEPHi DNNDK | DEEPHi DPU

- Compression
  - Pruning
  - Quantization
- Compilation
  - Compiler
  - Assembler
- Runtime
  - Core API
  - Loader
  - Driver
  - Profiler

AI HW Platforms

Z7020 Board | Z7020 SOM | ZU2 SOM | ZU2/3 Card | ZU9 Card | ZCU102 | ZCU104 | Ultra96

DNNDK Overview

> DECENT (DEep ComprEssioN Tool)

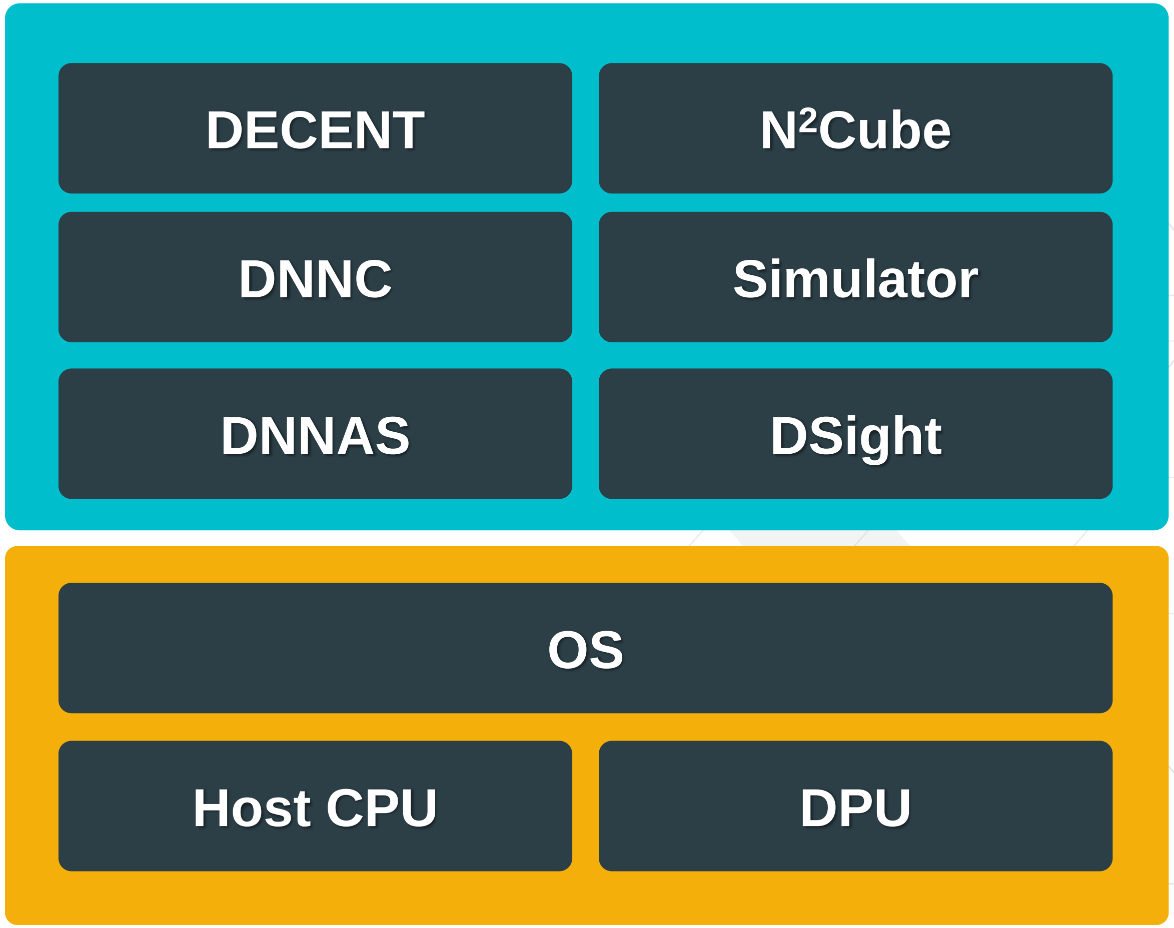
> DNNC (Deep Neural Network Compiler)

> DNNAS (Deep Neural Network ASsembler)

> Runtime N^2Cube (Cube of Neural Network)

> DPU Simulator

> Profiler DSight



Framework Support

Caffe

- Pruning tool @Caffe
- Quantization tool @Caffe

Darknet

- Pruning tool@darknet
- Quantization tool@darknet
- Convertor for caffe deploy
- Yolo V2 compression

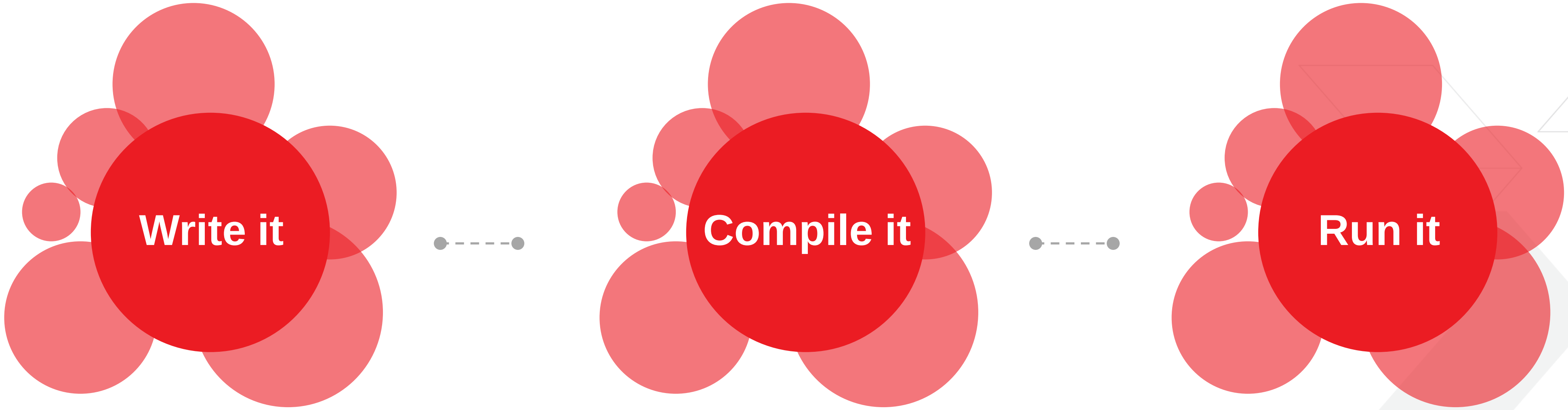
Tensorflow

(Coming soon)

- Quantization tool
- Pruning tool

Using DNNK in SDSoC

Only 3 steps!



Software defined development

Step 1 – Write It

These DNNDK APIs would wrap the C-callable IP ‘caller’ functions inside!

```
int main(ovid) {
    /* DPU Kernels/Tasks for running ResNet-50 */
    DPUKernel* kernelConv;
    DPUKernel* kernelFC;
    DPUTask* taskConv;
    DPUTask* taskFC;
    /* Attach to DPU driver and prepare for running*/
    dpuOpen();
    /* Create DPU kernels for CONV & FC Nodes in ResNet-50 */
    kernelConv = dpuLoadKernel(KERNEL_CONV);
    kernelFC = dpuLoadKernel(KERNEL_FC);
    /* Create DPU Tasks for CONV & FC Nodes in ResNet-50*/
    taskConv = dpuCreateTask(kernelConv, 0);
    taskFC = dpuCreateTask(kernelFC, 0);

    /* Run CONV & FC Kernels for ResNet-50 */
    runResnet50(taskConv, taskFC);

    /* Destroy DPU Task & free resources */
    dpuDestroyTask(taskConv);
    dpuDestroyTask(taskFC);
    /* Destroy DPU Kernels & free resources */
    dpuDestroyKernel(kernelConv);
    dpuDestroyKernel(kernelFC);
    /* Dettach from DPU driver & free resources */
    dpuClose();

    return 0;

}
```

ResNet-50 example

dpuOpen()	dpuGetOutputTensorSize()
dpuClose()	dpuGetOutputTensorScale()
dpuLoadKernel()	dpuGetOutputTensorHeight()
dpuDestroyKernel()	dpuGetOutputTensorWidth()
dpuCreateTask()	dpuGetOutputTensorChannel()
dpuRunTask()	dpuGetTensorSize()
dpuDestroyTask()	dpuGetTensorAddress()
dpuEnableTaskProfile()	dpuGetTensorScale()
dpuGetTaskProfile()	dpuGetTensorHeight()
dpuGetNodeProfile()	dpuGetTensorWidth()
dpuGetInputTensor()	dpuGetTensorChannel()
dpuGetInputTensorAddress()	dpuSetIntputTensorInCHWInt8()
dpuGetInputTensorSize()	dpuSetIntputTensorInCHWFP32()
dpuGetInputTensorScale()	dpuSetIntputTensorInHWCInt8()
dpuGetInputTensorHeight()	dpuSetIntputTensorInHWCFP32()
dpuGetInputTensorWidth()	dpuGetOutputTensorInCHWInt8()
dpuGetInputTensorChannel()	dpuGetOutputTensorInCHWFP32()
dpuGetOutputTensor()	dpuGetOutputTensorInHWCInt8()
dpuGetOutputTensorAddress()	dpuGetOutputTensorInHWCFP32()

DNNDK Programming C/C++ API

Step 2 – Compile It

Makefile

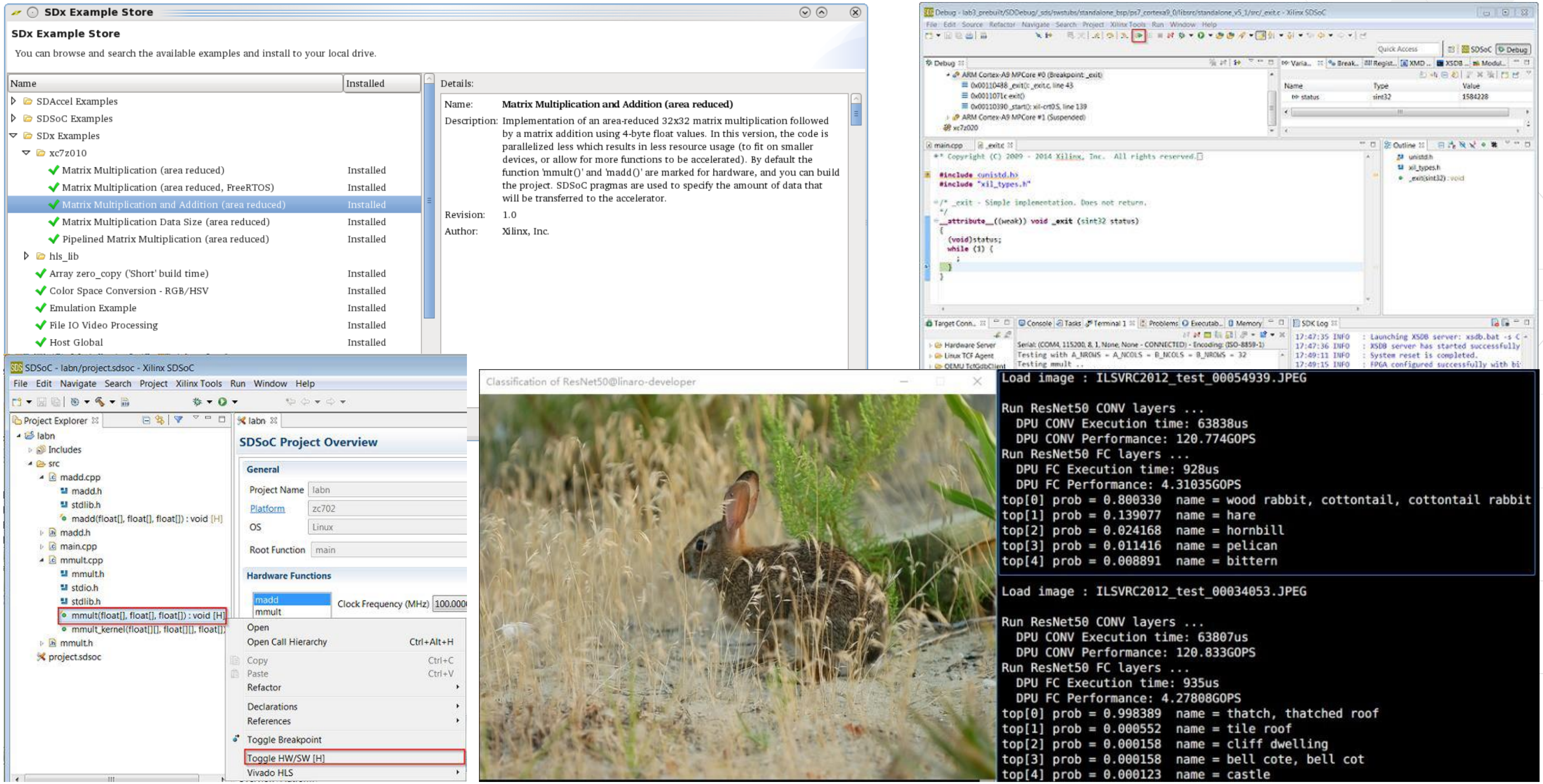
- sdscc/sds++
- -L./ -ldpu
- -sds-pf zcu102

SDx GUI

- Create project
- Set the flag
- Build it

Generate the bitstream, BOOT.BIN and .elf file

Step 3 – Run It


SDx Example Store
SDx Example Store
You can browse and search the available examples and install to your local drive.
Name
Installed
SDAccel Examples
SDSoC Examples
SDx Examples
xc7z010
Matrix Multiplication (area reduced) Installed
Matrix Multiplication (area reduced, FreeRTOS) Installed
Matrix Multiplication and Addition (area reduced) Installed
Matrix Multiplication Data Size (area reduced) Installed
Pipelined Matrix Multiplication (area reduced) Installed
hls_lib
Array zero_copy ('Short' build time) Installed
Color Space Conversion - RGB/HSV Installed
Emulation Example Installed
File IO Video Processing Installed
Host Global Installed
Details:
Name: Matrix Multiplication and Addition (area reduced)
Description: Implementation of an area-reduced 32x32 matrix multiplication followed by a matrix addition using 4-byte float values. In this version, the code is parallelized less which results in less resource usage (to fit on smaller devices, or allow for more functions to be accelerated). By default the function 'mmult()' and 'madd()' are marked for hardware, and you can build the project. SDSoC pragmas are used to specify the amount of data that will be transferred to the accelerator.
Revision: 1.0
Author: Xilinx, Inc.
SDSoC - labn/project.sdsoc - Xilinx SDSoC
File Edit Navigate Search Project Xilinx Tools Run Window Help
Project Explorer
labn
Includes
src
madd.cpp
madd.h
stdlib.h
madd(float[], float[], float[]) : void [H]
madd.h
main.cpp
mmult.cpp
mmult.h
stdio.h
stdlib.h
mmult(float[], float[], float[]) : void [H]
mmult_kernel(float[][], float[][], float[][])
mmult.h
project.sdsoc
labn
SDSoC Project Overview
General
Project Name labn
Platform zc702
OS Linux
Root Function main
Hardware Functions
madd
mmult
Clock Frequency (MHz) 100.000
Open
Open Call Hierarchy Ctrl+Alt+H
Copy Ctrl+C
Paste Ctrl+V
Refactor
Declarations
References
Toggle Breakpoint
Toggle HW/SW [H]
Vivado HLS
Classification of ResNet50@linaro-developer
Load image : ILSVRC2012_test_00054939.JPEG
Run ResNet50 CONV layers ...
DPU CONV Execution time: 63838us
DPU CONV Performance: 120.774GOPS
Run ResNet50 FC layers ...
DPU FC Execution time: 928us
DPU FC Performance: 4.31035GOPS
top[0] prob = 0.800330 name = wood rabbit, cottontail, cottontail rabbit
top[1] prob = 0.139077 name = hare
top[2] prob = 0.024168 name = hornbill
top[3] prob = 0.011416 name = pelican
top[4] prob = 0.008891 name = bittern
Load image : ILSVRC2012_test_00034053.JPEG
Run ResNet50 CONV layers ...
DPU CONV Execution time: 63807us
DPU CONV Performance: 120.833GOPS
Run ResNet50 FC layers ...
DPU FC Execution time: 935us
DPU FC Performance: 4.27808GOPS
top[0] prob = 0.998389 name = thatch, thatched roof
top[1] prob = 0.000552 name = tile roof
top[2] prob = 0.000158 name = cliff dwelling
top[3] prob = 0.000158 name = bell cote, bell cot
top[4] prob = 0.000123 name = castle

A Long Time for Every Build?

No! Once for all

SdSoC's system optimizing compiler will compare the new data motion network with the previous one. If they are the same, then there is no need to regenerate the Vivado project.

Each subsequent build will only take a few seconds. However, you need to make sure to:

- **Use the same C-callable IP library**
- **Use the same platform**
- **Use the same project settings**
- **At least call dpuOpen() in the code**

```
Generating data motion network
INFO: [DMAnalysis 83-4494] Analyzing hardware accelerators...
INFO: [DMAnalysis 83-4497] Analyzing callers to hardware accelerators...
INFO: [DMAnalysis 83-4444] Scheduling data transfer graph for partition 0
INFO: [DMAnalysis 83-4446] Creating data motion network hardware for partition 0
INFO: [DMAnalysis 83-4448] Creating software stub functions for partition 0
INFO: [DMAnalysis 83-4450] Generating data motion network report for partition 0
INFO: [DMAnalysis 83-4454] Rewriting caller code
Skipping block diagram (BD), address map, port information and device registration for partiti
on 0
Rewrite caller functions
```

Vision Libraries

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 the board

OpenCV

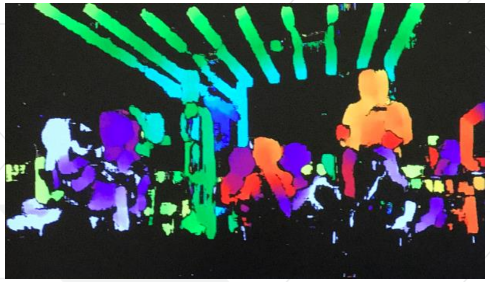
```
main(){
cv::imread(A);
cv::stereoRectify(A,B,C,D);
cv::stereoLBM(C,D,out);
cv::imshow(out);
}
```

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);
}
```



XDF XILINX DEVELOPER FORUM

XILINX

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	Hough Lines		
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			

Custom CV Function / Library Creation Flow

1 Cross-compile

2 Write custom CV function in C, C++

3 Assign functions to hardware. Compile using SDSoC

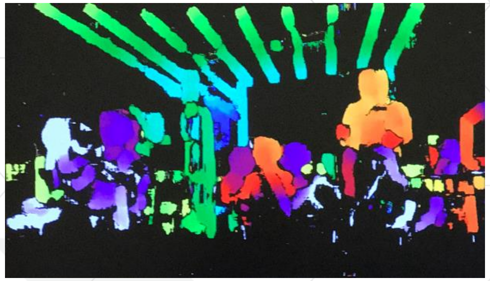
4 Run on the board

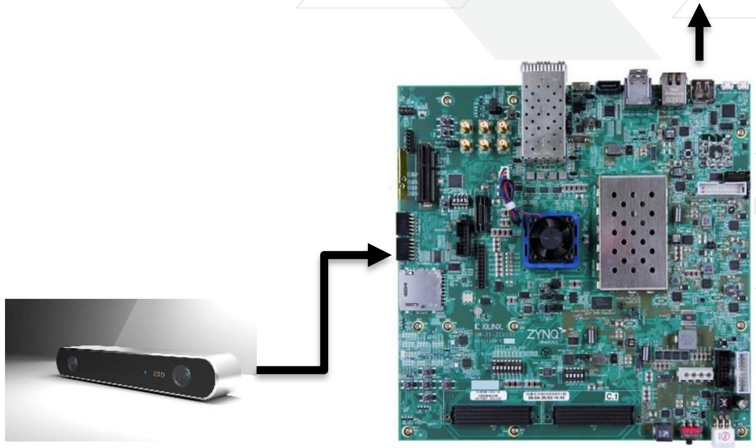
```
main(){
cv::imread(A);
xF:stereoRectify<line>(A,B,C,D);
xF:stereoLBM<win,n_disp>(C,D,E);
CUSTOM_CV(E,out);
cv::imshow(out);
}
```

```
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



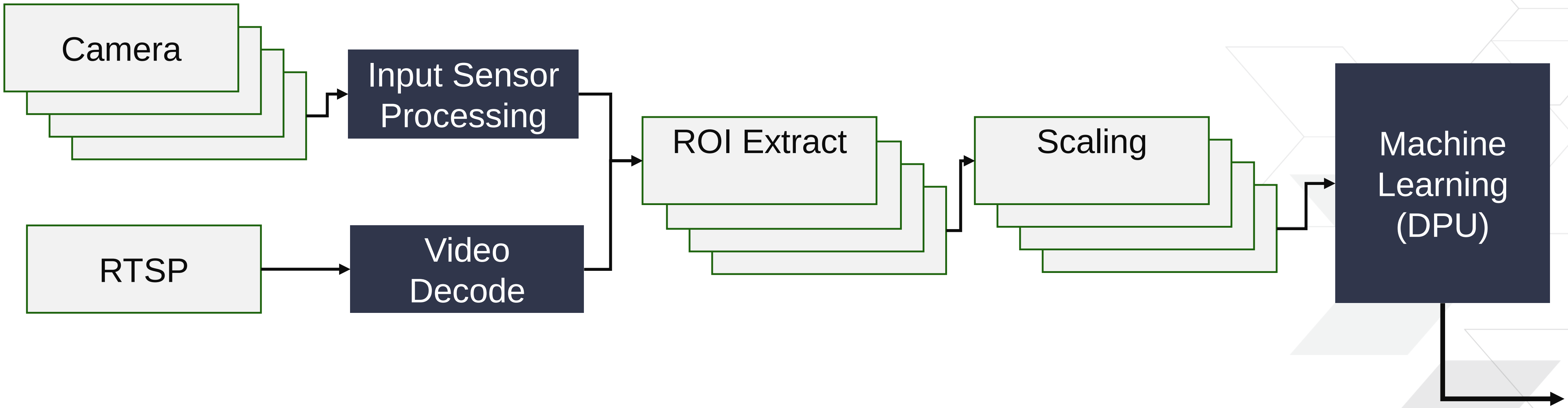


Building a Real System

Building a Real System

> **Components are great, but you still need to tie them together to build something!**

> **Real block diagrams are usually not at all simple…**

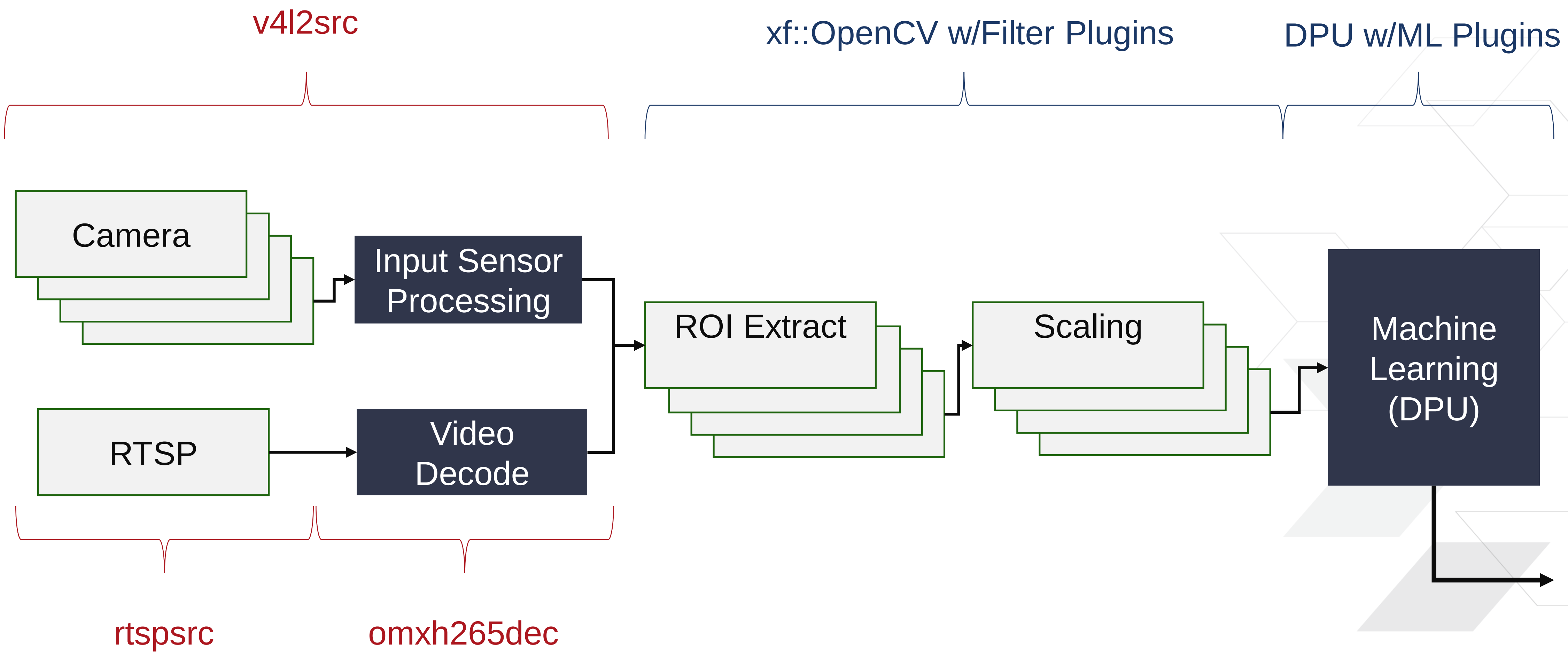


> **How do we easily tie all of this together? Move data efficiently?**

Frameworks!

- Flexible frameworks for combining video plugins, etc. already exist
  - Gstreamer is an extremely common open-source framework
- Gstreamer allows you to build flexible, graph-based representations of data transfers in video systems with arbitrary topologies
  - Handles buffer allocation and management, pipeline configuration, etc.
  - Xilinx has developed plugins (and plugin templates) for hardware and IP video functions along with accelerators
- Use hardware that's available, such as the VCU, HDMI, Ethernet, etc. with pre-provided plugins
- Then, use a combination of C-callable hardware libraries like xf::OpenCV and your own custom algorithms to glue everything together
- Take advantage of off-the-shelf C-callable libraries for machine learning and pre-traing, pre-optimized networks – or create your own!

With the Framework



Do I have to use Gstreamer?

> **No!**

> **Frameworks like gstreamer make building complex topologies easier, but you don't have to use them**

> **You are completely free to adapt these resources to your own topologies with your own custom software, frameworks, and libraries**

> **Our goal is to enable flexible, adaptable embedded vision systems leveraging machine learning and embedded vision processing**

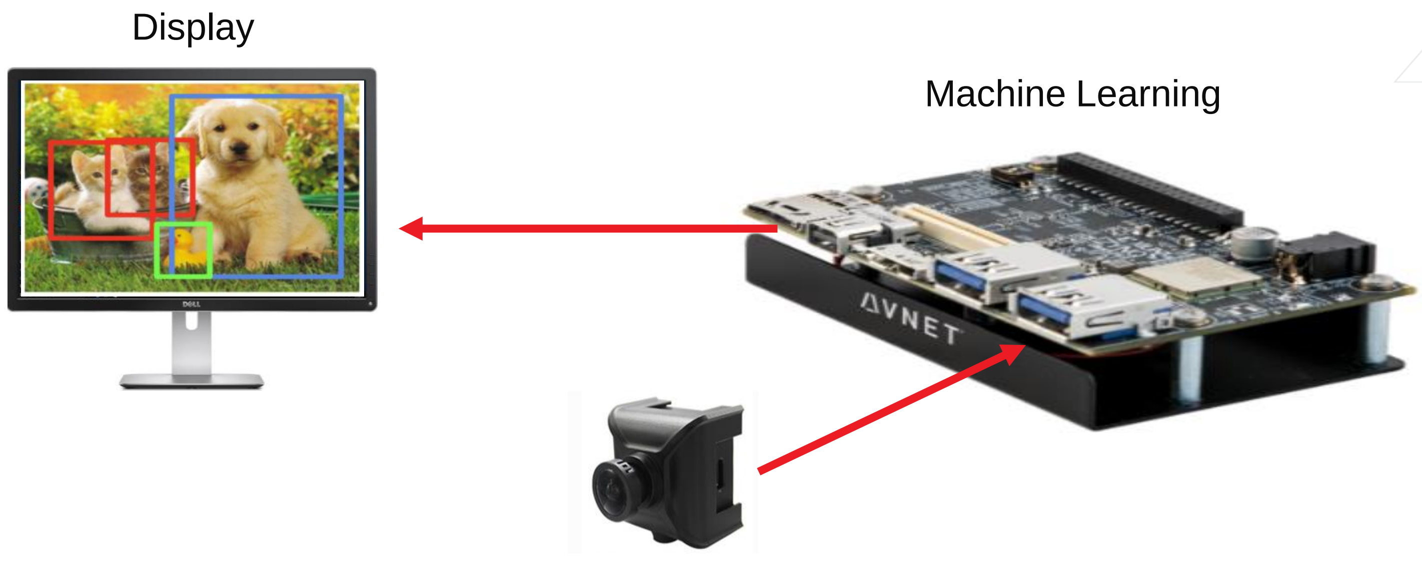
> **Low-level interactions with APIs for video, especially under Linux, can be complex**

>> Even if you don't intend to use them in a final design, tools like gstreamer can help you get started quickly!

Summary

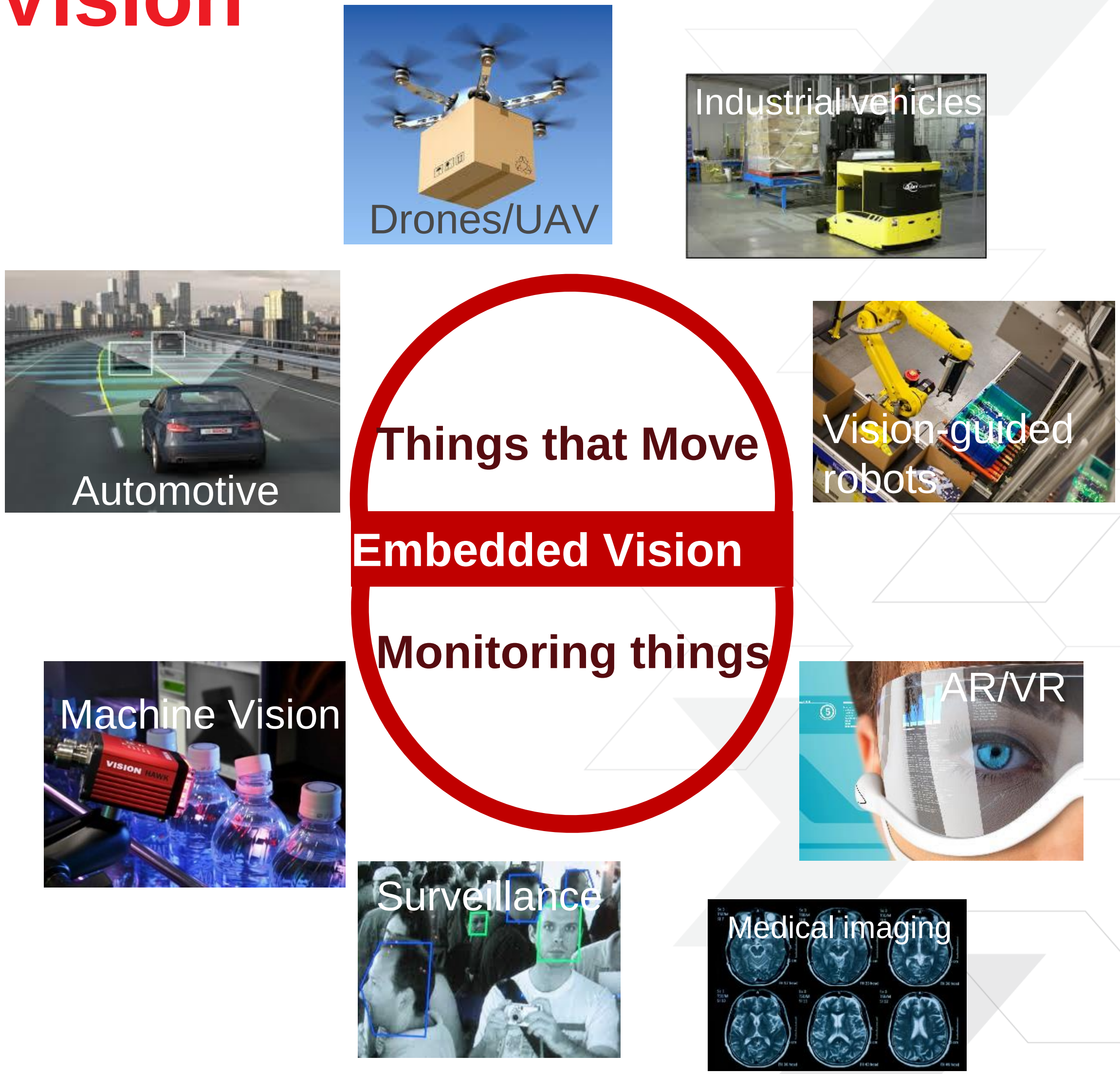
Xilinx Makes Adaptable ML Video Systems Easy

- **In the accompanying workshop session, we go from a design template to a working machine learning dinference design in roughly an hour**
  - Not, of course, including network training which is more time consuming
- **You can add ML processing to your own designs with relative ease using C-Callable IP Libraries in SDSoC**
- **Leverage gstreamer plugins to enable configurable vision pipelines**



Xilinx Focus on Embedded Vision

- **Artificial intelligence and embedded vision are rapidly changing the world**
- **Designers must be agile, adapting to evolving technologies quickly**
- **Xilinx provides a significant advantage with easy to use, powerful tools:**
  - Deep-learning
  - Accelerated software libraries
  - Sensor Boards and Kits



Diverse set of applications, only limited by your imagination

Call to Action

Evaluate SDSoC

> SDSoC is available today from Xilinx
https://github.com/Xilinx/SDSoC-Tutorials

Explore Machine Learning

> Contact your Xilinx representative for more information on edge and cloud machine learning IP, reference designs, and development kits

Explore Computer Vision

> Hardware-accelerated OpenCV functions are available on GitHub today!
https://github.com/Xilinx/xfopencv

XDF XILINX DEVELOPER FORUM