SDSoC, ML, and Embedded Vision

Presented By

Rob Armstrong
Sr. Product Marketing Manager, Xilinx, Inc.
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?
Frameworks & Libraries
- OpenCV
- gstreamer
- Machine Learning

Development tools
- SDSoC

Platforms
- USB3
- HDMI
- MIPI

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: XIL), the leader in adaptive and intelligent computing, announced today that it has acquired DeePhi Tech, a Beijing-based privately held startup with industry-leading capabilities in machine learning, specializing in deep compression, pruning, and system-level optimization for neural networks.
# Integrated Xilinx-Deephi Roadmap

## Xilinx AI Development

<table>
<thead>
<tr>
<th>Edge/Embedded</th>
<th>Cloud/DC</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>20+ pruned / customized / basic models</strong></td>
<td></td>
</tr>
</tbody>
</table>

### Models

- **Deephi Pruning**
  - Deephi Quantizer
  - Deephi Compiler
  - Deephi Runtime

### Software Stack

- **SDSoC**
  - Deephi Quantizer
- **SDAccel**
  - xFDNN Compiler
  - xFDNN Runtime

### FPGA IP

- **Deephi DPU**
- **Deephi LSTM**
- **xFDNN**

### Platforms

- **Z7020 Board**
- **Z7020 SOM**
- **ZU2/3 SOM**
- **ZU2/3 Card**
- **ZU9 Card**
- **ZCU102**
- **ZCU104**
- **Ultra96**

- **Xilinx U200, U250, U280**

---

© Copyright 2018 Xilinx

Xilinx - Deephi Roadmap

- Integrated Platforms
  - Z7020 Board
  - Z7020 SOM
  - ZU2/3 SOM
  - ZU2/3 Card
  - ZU9 Card
  - ZCU102
  - ZCU104
  - Ultra96

- Xilinx U200, U250, U280
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

C-callable IP

Optimized RTL IP

main()
{
    init(A,B,C);
    mmult(A,B,D);
    madd(C,D,E);
    consume(E);
}

result(inA,inB,out){

}

Synthesizable C/C++

HLS cross-compiled

System optimizing

Compiler infers data communication, uses data mover IP

Platform (can incl. PL)

OS, BSP

Target Board
SDSoC Tool Flow

C-callable IP

Optimized RTL IP

main() {
  init[A,B,C];
  mmult(A,B,D);
  madd(C,D,E);
  consume(S);
}

C/C++

Synthesizable

HLS cross-compiled

Compiler infers data communication, uses data mover IP

System linker generates hardware system, software stubs, APIs and configuration

Platform (can incl. PL)
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

Surveillance  ADAS/AD  Robotics  Data Center

And there are many more …
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
- DPU

AI HW Platforms

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

Compression

- Pruning
- Quantization

Compilation

- Compiler
- Assembler

Runtime

- Core API
- Loader
- Driver
- Profiler
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!

Write it

Compile it

Run it

Software defined development
Step 1 – Write It

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

```c
int main(oid) {
/* 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);
/* Detach from DPU driver & free resources */
dpuClose();
return 0;
}
```

ResNet-50 example

DNNDK Programming C/C++ API

- `dpuOpen()`
- `dpuClose()`
- `dpuLoadKernel()`
- `dpuDestroyKernel()`
- `dpuCreateTask()`
- `dpuRunTask()`
- `dpuDestroyTask()`
- `dpuEnableTaskProfile()`
- `dpuGetTaskProfile()`
- `dpuGetNodeProfile()`
- `dpuGetInputTensor()`
- `dpuGetInputTensorAddress()`
- `dpuGetInputTensorSize()`
- `dpuGetInputTensorScale()`
- `dpuGetInputTensorHeight()`
- `dpuGetInputTensorWidth()`
- `dpuGetInputTensorChannel()`
- `dpuGetOutputTensor()`
- `dpuGetOutputTensorAddress()`
- `dpuGetOutputTensorSize()`
- `dpuGetOutputTensorScale()`
- `dpuGetOutputTensorHeight()`
- `dpuGetOutputTensorWidth()`
- `dpuGetOutputTensorChannel()`
- `dpuGetTensorSize()`
- `dpuGetTensorAddress()`
- `dpuGetTensorScale()`
- `dpuGetTensorHeight()`
- `dpuGetTensorWidth()`
- `dpuGetTensorChannel()`
- `dpuSetInputTensorInCHWInt8()`
- `dpuSetInputTensorInCHWFP32()`
- `dpuSetInputTensorInHWCInt8()`
- `dpuSetInputTensorInHWCFP32()`
- `dpuGetOutputTensorInCHWInt8()`
- `dpuGetOutputTensorInCHWFP32()`
- `dpuGetOutputTensorInHWCInt8()`
- `dpuGetOutputTensorInHWCFP32()`
Step 2 – Compile It

Makefile
- sdscc/sds++
- -L./ -ldpu
- -sds-pfu zcu102

SDx GUI
- Create project
- Set the flag
- Build it

Generate the bitstream, BOOT.BIN and .elf file
Step 3 – Run It
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 partition 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

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

Cross-compile OpenCV application to Zynq (ARM A9/A53)

Profile and identify bottleneck functions

Minimal changes to the code and set functions to hardware.
Compile using SDSoC

Run on the board

<table>
<thead>
<tr>
<th>Name</th>
<th>Clock Frequency (MHz)</th>
</tr>
</thead>
<tbody>
<tr>
<td>stereoRectify</td>
<td>300</td>
</tr>
<tr>
<td>stereoLBM</td>
<td>300</td>
</tr>
</tbody>
</table>

```
main()
{
    cv::imread(A);
    xF: stereoRectify<line>(A,B,C,D);
    xF: stereoLBM<win,n_disp>(C,D,out);
    cv::imshow(out);
}
```
# xfOpenCV: 50+ Most Needed OpenCV Functions

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Clock Frequency (MHz)</th>
</tr>
</thead>
<tbody>
<tr>
<td>stereoRectify</td>
<td>300</td>
</tr>
<tr>
<td>stereoLBM</td>
<td>300</td>
</tr>
<tr>
<td>CUSTOM_CV</td>
<td>300</td>
</tr>
</tbody>
</table>
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-training, pre-optimized networks – or create your own!
With the Framework

Camera

Input Sensor Processing

ROI Extract

Scaling

Machine Learning (DPU)

v4l2src

rtspsrc

omxh265dec

xf::OpenCV w/Filter Plugins

DPU w/ML Plugins
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 inference 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