How to Accelerate OpenCV Applications with the Zynq-7000 All Programmable SoC using Vivado HLS Video Libraries

Dec 19, 2013
ZYNQ Overview

PS (Dual ARM Cortex™-A9 MPCore™)
- Operation clock: 776MHz @ speed 2
- NEON FPU/Vector SP/DP FP
- Hard macro Peripherals

PL (Artix7/Kintex 7)
- 28nm HPL process
- GTX 10.3125Gbps@ speed 2

- AXI Interconnect btw PS and PL
  - ACP
  - HP Port
  - GP Port

- Reduce BOM Cost – All-in one solution
  - Reduce System total power
  - Reduce PCB Layer
Vivado High-Level Synthesis (HLS)

C, C++ or SystemC

Vivado™ HLS

VHDL or Verilog

System IP Integration

Algorithmic Specification

Micro Architecture Exploration

RTL Implementation

Comprehensive Integration with the Xilinx Design Environment

Accelerates Algorithmic C to RTL IP integration
Benefits

Portability/Design Reuse

- Technology migration
- Cost reduction
- Full Resource/Performance/Power analysis

Design and IP reuse
Benefits

QoR (Quality of Results)
- Architecture Exploration
  - Timing
    - Parallelization
    - Pipelining
  - Resources
    - Sharing
- Heuristics

Rapid design exploration delivers QoR rivaling hand-coded RTL
Vivado High-Level Synthesis (HLS)

Accelerate Algorithmic C to Co-Processing Accelerator Integration
OpenCV Overview

- Open Source Computer Vision (OpenCV) is widely used to develop Computer Vision applications
  - Library of 2500+ optimized video functions
  - Optimized for desktop processors and GPUs
  - Tens of thousands users
  - Runs out of the box on ARM processors in Zynq

- However
  - HD processing with OpenCV is often limited by external memory
  - Memory bandwidth is a bottleneck for performance
  - Memory accesses limit power efficiency

- Zynq All-programmable SOCs are a great way of implementing embedded computer vision applications
  - High performance and Low Power
Real-Time Computer Vision Applications

**Computer Vision Applications**
- Advanced Drivers Assist for Safety
- Surveillance for Security
- Machine Vision for Quality
- Medical Imaging For non invasive surgery

**Real-time Analytics Function**
- Lane or Pedestrian detection
- Friend vs Foe recognition
- High velocity object detection
- Tumor detection

© Copyright 2013 Xilinx
Image Processing Example: Driver Assistance

- Analyze a video frame to detect road lane markings

- RGB to Gray Conversion
- Image Histogram Equalization
- Edge Detection
- Hough Transform

- Lane Detection
- Lane Tracking
- Decision Making
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
- Pixel Processing: Data or Frame Processing

Optimal Solution is a Mix of Hardware and Software
FPGAs allow massively parallel DSP processing

Standard DSP processor – Sequential (generic DSP)

- Data In
- Coefficients
- 864 clock cycles needed
- Single-MAC Unit
- Reg
- Data Out

\[
\frac{1.2 \text{ GHz}}{864 \text{ clock cycles}} = 1.38 \text{ MSPS}
\]

FPGA - Fully Parallel Implementation

- Data In
- C0, C1, C2, C3, ..., C640
- Reg
- 864 operations in 1 clock cycle
- Reg
- Data Out

\[
\frac{600 \text{ MHz}}{1 \text{ clock cycle}} = 600 \text{ MSPS}
\]

A 864-tap filter implementation is up to 432 times faster
Real-time Video Analytics Processing

Pixel based Image Processing and Feature Extraction

FullHD
- 1920x1080 ≈ 2K x 1K = 2M pixel
- 60 frame/sec
- 60 x 2M = 120M pixel / sec
- 100 operations / pixel
- 100x120M = 12G operations/sec

Typical embedded processor
- 1G ~ 10G ops

UHD
- 3840x2160 ≈ 4K x 2K = 8M pixel
- 7680x4320 ≈ 8K x 4K = 32M pixel
- 500 operations / pixel
- 500x120M = 60G operations/sec
Real-time Video Analytics Processing

Pixel based Image Processing and Feature Extraction

Frame based Feature processing and decision making

4Kx2K

100s Ops/pixel
8MPx100 Ops/frame = 100s Gops

F1
F2
F3
......

10000s Ops/feature
1000s of features/sec = Mops

© Copyright 2013 Xilinx
Heterogeneous Implementation of Real-time Video Analytics

Pixel based Image Processing and Feature Extraction

Frame based Feature processing and decision making

Hardware Domain (FPGA)

Software Domain (ARM)

100s Ops/pixel
8MPx100 Ops/frame
= 100s Gops

10000s Ops/feature
1000s of features/sec
= Mops
Xilinx Real-time Image Analytics Implementation: Zynq All Programmable SoC

Pixel based Image Processing and Feature Extraction

Frame based Feature processing and decision making

4Kx2K

100s Ops/pixel
8MPx100 Ops/frame = 100s Gops

10000s Ops/feature
1000s of features/sec = Mops

© Copyright 2013 Xilinx
Accelerating OpenCV Applications

- Driver Assist
- Broadcast Monitor
- HD Surveillance
- Cinema Projection
- Video Conferencing
- Digital Signage
- Studio Cinema Camera
- Consumer Displays
- Office-class MFP
- Medical Displays
- Machine Vision

Frame-level processing Library for PS
Pixel processing interfaces and basic functions for analytics
Vivado HLS

© Copyright 2013 Xilinx
Vivado: Productivity gains for OpenCV functions

- C simulation of HD video algorithm ~1 fps
  - RTL simulation of HD video 1 frame per hour

- Real-time FPGA implementation up to 60fps
Video Library Functions

- C++ code contained in hls namespace.
  - #include "hls_video.h" & "hls_opencv.h"

- Similar interface, equivalent behavior with OpenCV, e.g.
  - 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, e.g.
  - 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
### Video Library Core Structures

<table>
<thead>
<tr>
<th>OpenCV</th>
<th>HLS Video Library</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>cv::Point_&lt;T&gt;, CvPoint</code></td>
<td><code>hls::Point_&lt;T&gt;, hls::Point</code></td>
</tr>
<tr>
<td><code>cv::Size_&lt;T&gt;, CvSize</code></td>
<td><code>hls::Size_&lt;T&gt;, hls::Size</code></td>
</tr>
<tr>
<td><code>cv::Rect_&lt;T&gt;, CvRect</code></td>
<td><code>hls::Rect_&lt;T&gt;, hls::Rect</code></td>
</tr>
<tr>
<td><code>cv::Scalar_&lt;T&gt;, CvScalar</code></td>
<td><code>hls::Scalar&lt;N, T&gt;</code></td>
</tr>
<tr>
<td><code>cv::Mat, IplImage, CvMat</code></td>
<td><code>hls::Mat&lt;ROWS, COLS, T&gt;</code></td>
</tr>
<tr>
<td><code>cv::Mat mat(rows, cols, CV_8UC3);</code></td>
<td><code>hls::Mat&lt;ROWS, COLS, HLS_8UC3&gt; mat (rows, cols);</code></td>
</tr>
<tr>
<td><code>IplImage* img = cvCreateImage(cvSize(cols,rows), IPL_DEPTH_8U, 3);</code></td>
<td><code>hls::Mat&lt;ROWS, COLS, HLS_8UC3&gt; img, (rows, cols);</code></td>
</tr>
<tr>
<td></td>
<td><code>hls::Mat&lt;ROWS, COLS, HLS_8UC3&gt; img;</code></td>
</tr>
<tr>
<td></td>
<td><code>hls::Window&lt;ROWS, COLS, T&gt;</code></td>
</tr>
<tr>
<td></td>
<td><code>hls::LineBuffer&lt;ROWS, COLS, T&gt;</code></td>
</tr>
</tbody>
</table>
HLS Video Libraries

- OpenCV functions are not directly synthesizable with HLS
  - Dynamic memory allocation
  - Floating point
  - Assumes images are modified in external memory

- The HLS video library is intended to replace many basic OpenCV functions
  - Similar interfaces and algorithms to OpenCV
  - Focus on image processing functions implemented in FPGA fabric
  - Includes FPGA-specific optimizations
    - Fixed point operations instead of floating point
    - On-chip Linebuffers and window buffers
  - Not necessarily bit-accurate

- HLS Video Libraries
  - Data Type
  - Memory Window
  - Memory Line Buffer
  - Video Functions
# Xilinx HLS Video Library 2013.2

## Video Data Modeling

<table>
<thead>
<tr>
<th>Linebuffer class</th>
<th>Window class</th>
</tr>
</thead>
</table>

## AXI4-Stream IO Functions

<table>
<thead>
<tr>
<th>AXIvideo2Mat</th>
<th>Mat2AXIvideo</th>
</tr>
</thead>
</table>

## OpenCV Interface Functions (OpenCV data type <-> AXI-4 Stream)

<table>
<thead>
<tr>
<th>cvMat2AXIvideo</th>
<th>AXIvideo2cvMat</th>
</tr>
</thead>
<tbody>
<tr>
<td>IplImage2AXIvideo</td>
<td>AXIvideo2IplImage</td>
</tr>
</tbody>
</table>

## CVMat2AXIvideo | AXIvideo2CvMat |

## Video Functions

<table>
<thead>
<tr>
<th>AbsDiff</th>
<th>Duplicate</th>
<th>MaxS</th>
<th>Remap</th>
</tr>
</thead>
<tbody>
<tr>
<td>AddS</td>
<td>EqualizeHist</td>
<td>Mean</td>
<td>Resize</td>
</tr>
<tr>
<td>AddWeighted</td>
<td>Erode</td>
<td>Merge</td>
<td>Scale</td>
</tr>
<tr>
<td>And</td>
<td>FASTX</td>
<td>Min</td>
<td>Set</td>
</tr>
<tr>
<td>Avg</td>
<td>Filter2D</td>
<td>MinMaxLoc</td>
<td>Sobel</td>
</tr>
<tr>
<td>AvgSdv</td>
<td>GaussianBlur</td>
<td>MinS</td>
<td>Split</td>
</tr>
<tr>
<td>Cmp</td>
<td>Harris</td>
<td>Mul</td>
<td>SubRS</td>
</tr>
<tr>
<td>CmpS</td>
<td>HoughLines2</td>
<td>Not</td>
<td>SubS</td>
</tr>
<tr>
<td>CornerHarris</td>
<td>Integral</td>
<td>PaintMask</td>
<td>Sum</td>
</tr>
<tr>
<td>CvvtColor</td>
<td>InitUndistortRectifyMap</td>
<td>Range</td>
<td>Threshold</td>
</tr>
<tr>
<td>Dilate</td>
<td>Max</td>
<td>Reduce</td>
<td>Zero</td>
</tr>
</tbody>
</table>

> For function signatures and descriptions, see the HLS user guide UG902
Partitioned OpenCV Application

1) Develop OpenCV application on Desktop
2) Run OpenCV application on ARM cores without modification
3) Abstract FPGA portion using I/O functions
4) Replace OpenCV function calls with synthesizable code
5) Replace call to synthesizable code with call to FPGA accelerator
6) Run HLS to generate FPGA accelerator
OpenCV design flow

1) Develop OpenCV application on Desktop
2) Run OpenCV application on ARM cores without modification
3) Abstract FPGA portion using I/O functions
4) Replace OpenCV function calls with synthesizable code
5) Run HLS to generate FPGA accelerator
6) Replace call to synthesizable code with call to FPGA accelerator
Implementing an OpenCV System on Zynq

OpenCV Application

- Image Read (OpenCV)
- OpenCV function chain
- Image Write (OpenCV)

Video Library Application

- Image Read (OpenCV)
- OpenCV2AXIvideo
- AXIvideo2Mat
- HLS Video Library Function Chain
- Mat2AXIvideo
- AXIvideo2OpenCV
- Image Write (OpenCV)

Zynq Implementation

- Processing System
- Memory Interfaces
- ARM® Dual Cortex-A9 MPCore™ System
- Common Peripherals
- Custom Accelerators
- Common Peripherals
- Custom Peripherals
- 7 Series Programmable Logic

OpenCV Application Video Library Application Zynq Implementation

OpenCV Application Video Library Application Zynq Implementation

- Image Read (OpenCV)
- OpenCV2AXIvideo
- AXIvideo2Mat
- HLS Video Library Function Chain
- Mat2AXIvideo
- AXIvideo2OpenCV
- Image Write (OpenCV)

OpenCV Application Video Library Application Zynq Implementation

OpenCV Application Video Library Application Zynq Implementation

- Image Read (OpenCV)
- OpenCV2AXIvideo
- AXIvideo2Mat
- HLS Video Library Function Chain
- Mat2AXIvideo
- AXIvideo2OpenCV
- Image Write (OpenCV)
OpenCV Design Tradeoffs

- OpenCV-based image processing is built around memory frame buffers
  - Poor access locality -> small caches perform poorly
  - Complex architectures for performance -> higher power
  - Likely ‘good enough’ for many applications
    - Low resolution or framerate
    - Processing of features or regions of interest in a larger image

- Streaming architectures give high performance and low power
  - Chaining image processing functions reduces external memory accesses
  - Video-optimized line buffers and window buffers simpler than processor caches
  - Can be implemented with streaming optimizations in HLS
  - Requires conversion of code to be synthesizable
Limitations

- Must replace OpenCV calls with video library functions
- Frame buffer access not supported through pointers
  - use VDMA and AXI Stream adapter functions
- Random access not supported
  - data read more than once must be duplicated
  - see hls::Duplicate()
- In-place update not supported
  - e.g. cvRectangle (img, point1, point2)

### OpenCV

Read operation
```cpp
def read_opencv():
    pix = cv_mat.at<T>(i,j)
    pix = cvGet2D(cv_img, i, j)
```

Write operation
```cpp
def write_opencv():
    cv_mat.at<T>(i,j) = pix
    cvSet2D(cv_img, i, j, pix)
```

### HLS Video Library

Read operation
```cpp
def read_hls_video():
    hls_img >> pix
```

Write operation
```cpp
def write_hls_video():
    hls_img << pix
```
Vivado HLS System IP Integration Flow

C based IP Creation

- C, C++ or SystemC

C Libraries
- math.h
- Video
- DSP (2H13)

Vivado™ HLS

VHDL or Verilog & SW Driver

User Preferred System Integration Environment

System Generator for DSP

Vivado IP Integrator

Vivado RTL Integration

IP Catalog

7 Series FPGA and Zynq SoC Vivado Implementation
Zynq Video TRD architecture

- Video access to external memory using 64-bit High Performance ports
- Control register access using 32-bit General Purpose ports
- Video streams implemented using AXI4-Stream
Using OpenCV in FPGA designs

Pure OpenCV Application

- Image File Read (OpenCV)
- OpenCV function chain
- Image File Write (OpenCV)

Integrated OpenCV Application

- Live Video Input
- OpenCV function chain
- Live Video Output

OpenCV Reference

- Image File Read (OpenCV)
- OpenCV2AXIvideo
- AXIvideo2Mat
- HLS video library function chain
- Mat2AXIvideo
- AXIvideo2OpenCV
- Image File Write (OpenCV)

Accelerated OpenCV Application

- Live Video Input
- AXIvideo2Mat
- HLS video library function chain
- Mat2AXIvideo
- Live Video Output

© Copyright 2013 Xilinx
Pure OpenCV Application

Image File Read (OpenCV)

OpenCV function chain

Image File Write (OpenCV)
Integrated OpenCV Application

Live Video Input
- OpenCV function chain
- Live Video Output

Processing System
- SD Card
- Hardened Peripherals
- DDR Memory Controller
- Dual Core Cortex-A9
- AXI Interconnect
- AXI VDMA
- HLS-generated pipeline
- Xylon Display Controller
- DDR3 External Memory

Video Input

HDMI

© Copyright 2013 Xilinx
OpenCV Reference / Software Execution

Image File Read (OpenCV)
- OpenCV2AXIvideo
- AXIvideo2Mat
- HLS video library function chain
- Mat2AXIvideo
- AXIvideo2OpenCV
- Image File Write (OpenCV)
OpenCV Reference / In system Test

- Image File Read (OpenCV)
- OpenCV2AXIvideo
- AXIvideo2Mat
- HLS video library function chain
- Mat2AXIvideo
- AXIvideo2OpenCV
- Image File Write (OpenCV)

System Components:
- SD Card
- DDR3 External Memory
- Processing System
- AXI VDMA
- HDMI Video Input

Interconnections:
- DDR Memory Controller
- Dual Core Cortex-A9
- HLS-generated pipeline
- Xylon Display Controller
- AXI Interconnect
Accelerated OpenCV Application

Live Video Input
- AXIvideo2Mat
- HLS video library function chain
- Mat2AXIvideo

Live Video Output

AXI Interconnect

Processing System
- SD Card
- Hardened Peripherals

DDR Memory Controller
- Dual Core Cortex-A9

AXI VDMA
- HLS-generated pipeline

Video Input

Xylon Display Controller

HDMI Input

HDMI Output
Performance Analysis

- AXI Performance Monitor collects statistics on memory bandwidth
  - see /mnt/AXI_PerfMon.log
- Video + fast corners
  - 1920*1080*60*32 = ~4 Gb/s per stream
  - HP0: Read 4.01 Gb/s, Write 4.01 Gb/s, Total 8.03 Gb/s
  - HP2: Read 4.01 Gb/s, Write 4.01 Gb/s, Total 8.03 Gb/s
Voltage and Current can be read from the digital power regulators on the ZC702 board.

Custom, realtime HD video processing in 2-3 Watts total system power

- FASTX is less than 200 mW incremental power
HLS and Zynq accelerates OpenCV apps

- OpenCV functions enable fast prototyping of Computer Vision algorithms
- Computer Vision applications are inherently heterogenous and require a mix HW and SW implementation
- Vivado HLS video library accelerates mapping of openCV functions to FPGA programmable fabric
- Zynq offers power-optimized integrated solution with high performance programmable logic and embedded ARM
Download XAPP1167 from Xilinx.com

QuickTake: Leveraging OpenCV and High-Level Synthesis with Vivado

http://www.xilinx.com/hls
http://www.xilinx.com/getlicense