Programming vision applications on Zynq using OpenCV and High-Level Synthesis

Kees Vissers
MPSoC, July 2013

Video And Vision processing

From Pixels to information
### Required Pixel Rate Processing vs. Capabilities

<table>
<thead>
<tr>
<th>Video Format</th>
<th>Pixel rate M Pix/s</th>
<th>Per Pixel</th>
<th>Programmable Solution</th>
</tr>
</thead>
<tbody>
<tr>
<td>4K</td>
<td>480</td>
<td>240 Gops</td>
<td></td>
</tr>
<tr>
<td>1080p</td>
<td>120</td>
<td>60 Gops</td>
<td></td>
</tr>
<tr>
<td>720p</td>
<td>20</td>
<td>10 Gops</td>
<td></td>
</tr>
<tr>
<td>DVD</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>VCD</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Hi Res. Display Pixel Rate Processing Exceeds RISC Based Capabilities**

### Processes and Pipelines

<table>
<thead>
<tr>
<th>Design approach</th>
<th>RISC Proc.</th>
<th>Proc. w/ accels.</th>
<th>Folded datapath</th>
<th>Pipelined datapath</th>
<th>Replicated datapath</th>
</tr>
</thead>
<tbody>
<tr>
<td>clock:sample</td>
<td>1000:1</td>
<td>100:1</td>
<td>10:1</td>
<td>1:1</td>
<td>1:10</td>
</tr>
<tr>
<td>Data Rate (200MHz clock)</td>
<td>200Ks/s</td>
<td>2Ms/s</td>
<td>20Ms/s</td>
<td>200Ms/s</td>
<td>2 Gs/s</td>
</tr>
</tbody>
</table>

**Applications**

- control → audio → mobile video → HDTV → comms → networking

**FPGAs**
Zynq Products in context for video

Design approach

<table>
<thead>
<tr>
<th>Clock:Sample</th>
<th>RISC Proc.</th>
<th>Proc. w/ accels.</th>
<th>Folded datapath</th>
<th>Pipelined datapath</th>
<th>Replicated datapath</th>
</tr>
</thead>
<tbody>
<tr>
<td>1000:1</td>
<td>100:1</td>
<td>10:1</td>
<td>1:1</td>
<td>1:10</td>
<td></td>
</tr>
</tbody>
</table>

Data Rate (200MHz clock)

- 200Ks/s
- 2Ms/s
- 20Ms/s
- 200Ms/s
- 2 Gs/s

Applications
- Frame rate processing → line rate processing → HDTV pixel rate → ...

Arm A9 processors
- 1-2 Gops

Fabric
- 10 – 500 Gops

Zynq platform

Processing System

- Static Memory Controller
  - Quad-SPI, NAND, NOR
- Dynamic Memory Controller
  - DDR3, DDR2, LPDDR2

AMBA® Switches

ARM® CoreSight™ Multi-core & Trace Debug

NEON™/ FPU Engine

Cortex™-A9 MPCore™ 32/32 KB I/D Caches

512 KB L2 Cache

Snoop Control Unit (SCU)

Timer Counters

General Interrupt Controller

256 KB On-Chip Memory

DMA

Configuration

GPIO

2x SPI

2x I2C

2x CAN

2x UART

2x SDIO with DMA

2x USB with DMA

2x GigE with DMA

AMBA® Switches

Programmable Logic:
- System Gates, DSP, RAM

Multi-Standards I/Os (3.3V & High Speed 1.8V)

Multi Gigabit Transceivers

Multi-Standards I/Os (3.3V & High Speed 1.8V)

PCIe

XADC

Multi- Standards II/Os (3.3V & High Speed 1.8V)
Video Board: Zync 702 board

Programming

- Image processing often programmed using streaming and OpenCV libraries
- Processor runs Linux, and a window system (Qt)
- On chip performance monitors tied to running Display, processor load and FPGA to external memory load

Basic idea:
- take the edge detect of the current frame and
- the edge detect of previous frame
- Subtract: if the same: nothing, if different: show fat pixel
while(1){
  //Get the image frame from the video input
  frame_current = cvQueryFrame(capture);
  
  //Detect edges in the current frame
  cvSobel( gray_current, edge_current2, 1, 0, aperture_size );
  cvSobel( gray_current, edge_current1, 0, 1, aperture_size );
  cvAdd(edge_current2,edge_current1,edge_current2,NULL);
  cvConvertScale(edge_current2,edge_current,scale,0);
  cvThreshold(edge_current,edge_current,5,255,CV_THRESH_BINARY);

  //Detect edges in the previous frame
  cvSobel( gray_prev, edge_prev2,1, 0, aperture_size );
  cvSobel( gray_prev, edge_prev1,0, 1, aperture_size );
  cvAdd(edge_prev2,edge_prev1,edge_prev2,NULL);
  cvConvertScale(edge_prev2,edge_prev,scale,0);
  cvThreshold(edge_prev,edge_prev,5,255,CV_THRESH_BINARY);

  //Detect edges that are only present in the edge_current image
  detect_new_edges(edge_prev,edge_current,new_edge);

  //Remove noise from the new edges
  cvSmooth(new_edge,filtered_new_edges,CV_MEDIAN,7,7);

  //Combine new edges with current frame
  highlight_blend(frame_current,filtered_new_edges,output_frame);

  //Copy current frame into previous frame
  cvCopy(frame_current,frame_prev,NULL);

  //Display output frame
  cvShowImage("Detector Output",output_frame);
}
Current Frame

Previous Frame

OpenCV Motion Detection Algorithm

Detected New Car

External Input/Output and compute

HDMI to Framebuffer IO IP subsystem

HDMI to FB

sobel_filter_pass();
sobel_filter();
diff_image();
median_char_filter_pass();
combo_image();
ycbcr2rgb_pad();

FB to HDMI

HDMI Output

logicvc

Framebuffer to HDMI IO IP subsystem

• Frame Buffer is the application level abstraction for HDMI input/output

MPSoC 2013
Laptop demo with webcam

- Exactly the same OpenCV image processing pipe
- Using OpenCV libraries optimized for Intel SSE vector processing
- Webcam is 1280 x 720 (720p)
- Runs on Intel i7 with ~2.7GHZ processor and 8Gbyte DRAM
- Net result in the range of one frame every 1-2 seconds

Demo

Complete setup:

- Model train
- HDTV camera 1080p 60Frames per second, HDMI
- Board, with application running, linux, Qt, processor + Bus -load
- Mouse tied to a register to set threshold value dynamically
- HDTV, 1080p, HDMI
Power zones

- **I/O MUX**
- **MIO**
- **GPIO**
- **2x SPI**
- **2x I2C**
- **2x CAN**
- **2x UART**
- **2x USB with DMA**
- **2x GigE with DMA**

**Processing System**

- **Static Memory Controller**
  - Quad-SPI, NAND, NOR
- **Dynamic Memory Controller**
  - DDR3, DDR2, LPDDR2
- **AMBA® Switches**
- **ARM® CoreSight™ Multi-core & Trace Debug**
- **NEON™/ FPU Engine**
- **Cortex™-A9 MPCore™**
  - 32/32 KB I/D Caches
- **Snoop Control Unit (SCU)**
- **Timer Counters**
- **512 KB L2 Cache**
- **General Interrupt Controller**
- **NEON™/ FPU Engine**
- **Cortex™-A9 MPCore™**
  - 32/32 KB I/D Caches

**Programmable Logic:**

- **System Gates, DSP, RAM**
- **AMBA® Switches**
- **INT**
- **PINT**
- **AUX, ADJ**
- **BRAM**
- **AUX, ADJ**
- **PINT**
- **INT**

**Power zones**

- **1V5**
- **3V3**
- **2V5**
Power Measurement output on running board

- Power in Watts
- Only few measurements/sec
- 10% noise
- 10% difference between bit streams

Performance and Power Measurements

- All Pixel processing with OpenCV libraries running on ARM A9: one frame per 13 secs
- All Pixel processing with C++ running on A9: 1 frame per 1-2 seconds
- All Pixel processing with C++ libraries implemented via HLS in FPGA: 60 frames per second, FPGA runs 130MHz
- A9 processors: 500mW – 800mW
- FPGA fabric fully running: 500mW – 1W
- On Chip I/O few hundred mW, on board DRAM 800mW
- Result: ~ 100x speedup at SAME power consumption, ~100GOPS
- Energy efficiency is in the 100 – 200 Gops/W range for the FPGA in the complete system!
- You can put your finger on the running chip in the system, warm but not HOT : less than ~2 W!
Energy Efficiency (MOPS/mW or OP/nJ)

- Courtesy Bob Brodersen, based on published results at ISSCC conferences.

Energy Efficiency (MOPS/mW or GOPS/W)

- Courtesy Bob Brodersen, based on published results at ISSCC conferences.
Conclusion

- Every processor benefits from a combination with FPGA: Zynq device
- Every FPGA benefits from a combination with a processor: Zynq device
- We have shown an application programmed in OpenCV, leveraging Vivado HLS running on a 1-2W Zynq device at 1080p 60fps real-time.

Special thanks to the team that worked on the demo: Jack Lo, Fernando Martinez Vallina, S. Mohan, Vinod Kathail, and to many colleagues in the Vivado High Level Synthesis team and the Video Platform teams.

You can do this too:

- OpenCV and HLS video:
- OpenCV and HLS application note:
- Xilinx Zynq 702 board:
  http://www.zedboard.org/