A Software Developer's Journey into a Deeply Heterogeneous World

Tomas Evensen, CTO Embedded Software, Xilinx
Embedded Development: Then

- Simple single CPU
- Most code developed internally
  - 10’s of thousands of lines of code in C and assembly
- Single Real-time Operating System
- JTAG/BDM debugger
- Simple I/O
Embedded Development: Now

- Multiple heterogeneous CPUs
- Multiple accelerators and programmable logic
- Millions of lines of code - Mostly from other places like open source
- Multiple Operating Systems (i.e. Linux + RTOS)
- JTAG debugger
- Safety and Security concerns

Xilinx Zynq MPSoC

![Diagram of Xilinx Zynq MPSoC](image-url)
Dedicated Hardware is Energy Efficient

![Graph showing energy efficiency comparison between Microprocessors, General Purpose DSP, and Dedicated hardware. The graph indicates that Dedicated hardware is significantly more energy efficient with 3 orders of magnitude improvement compared to Microprocessors.]

Courtesy Bob Broderson, based on published results at ISSCC conferences.
Heterogeneous Example: IIoT Gateway

Expertise Needed All the Way from a System Level to Cloud Connectivity
FPGA – The “Chameleon” Chip

- Is it glue logic?
- Is it a powerful parallel DSP engine?
- Is it an RTL simulator?

Yes!!

And more…
FPGA – Reaching New Developers

- Limited pool of FPGA developers
  - Need to reach software developers
  - Software developers are different!

- Key to reach software developers
  1. Create libraries so they can utilize accelerators written by others
  2. Create tools so they can utilize FPGA without RTL
Heterogeneous Software Development
Mapping Applications to Heterogeneous Systems

![Diagram showing mapping of applications to heterogeneous systems](image-url)
Components for Heterogeneous SW Development

- Accelerated libraries and frameworks for common functions
  - E.g. OpenCV, CNN, ...

- Support for multiple types of Operating Environments
  - Solid Linux support, bare metal, FreeRTOS, 3rd party RTOS, Windows EC
  - Mixing of OS’s through AMP and hypervisors

- System debugger – Unifying debug/profile
  - Debug across cores and FPGA including profiling and trace

- FPGA Compiler – SDSoC
  - Write code for FPGA using C/C++/OpenCL
  - Automate the “glue” between execution engines

- Other
  - Virtual Prototyping for complete system
Framework Programming: Deep Learning

- Many embedded problems are being converted to use deep learning
  - Embedded vision, speech, …
  - Using neural networks of different kinds, e.g. CNN, …

- Neural networks are “programmed” through learning

- Neural networks are typically controlled by frameworks
  - Caffe, Tensorflow, Torch, Theano, …

- Neural networks are very computation intensive

- FPGAs can be very efficient for neural networks
  - Combination of fixed point, flexible routing, memory hierarchies and DSPs
  - By supporting existing framework, programmers can avoid RTL

<table>
<thead>
<tr>
<th>Output Feature Maps</th>
<th>Filter Sizing</th>
<th>MACs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rows</td>
<td>Cols</td>
<td>Depth</td>
</tr>
<tr>
<td>conv1</td>
<td>55</td>
<td>55</td>
</tr>
<tr>
<td>conv2</td>
<td>27</td>
<td>27</td>
</tr>
<tr>
<td>conv3</td>
<td>13</td>
<td>13</td>
</tr>
<tr>
<td>conv4</td>
<td>13</td>
<td>13</td>
</tr>
<tr>
<td>conv5</td>
<td>13</td>
<td>13</td>
</tr>
<tr>
<td>fc6</td>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td>fc7</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>fc8</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
OpenAMP: A Standard for Multi-OS Systems

What is OpenAMP?
- A standard for mixing embedded Operating Systems
- An Open Source project

Trend to combine Operating Systems
- Linux is used in majority of use cases
- Many free and commercial RTOS's are being used
- Bare metal (no OS) is common on smaller cores

Why multiple Operating Systems?
- Heterogeneous cores
- Different needs
  - Real-time vs. general purpose
  - Different Safety/Security levels
  - Legacy
  - GPL avoidance

Safety and Security issues common
- Affects boot order, messaging implementation, …
OpenAMP Capabilities

- **Provides a Layer for Applications**
  - Standard API’s that allow applications to be ported across processors and operating systems

- **System Development**
  - Provides a wide range of capabilities needed to deploy applications across asymmetric computing elements

- **Inter-OS & Inter Processor Communication**
  - Send messages back and forth

- **OS Management**
  - Provides booting/rebooting of processors

- **Two Implementations**
  - GPL implementation in Linux kernel
  - BSD implementation for RTOS/BM/Linux user space
Software Development Tools (SDK)

Complete system visibility needed
- Heterogeneous debugging and analysis is very hard!
- Especially timing related problems

Tools Features:
- Heterogeneous system level debugging
  - Visibility into both CPUs and FPGA
- Integrated performance profiling
  - Which parts of the chip are busy?
  - Measure processor and bus activities
  - Integrated traffic generator
- System event trace
  - What is happening in the chip over time?
  - Combined time line for SW and HW events
- Based on standards – Open source Eclipse, TCF

71% - Software Development tools

Strong system level tools are critical for heterogeneous development
Performance Data

Live tables
- ARM performance registers
  - Cache misses, IPC, ...
- AXI performance registers
  - Transactions, latency, ...
- Non-intrusive JTAG profiling

Timeline plot
- Correlate performance
  - Cache, busses, CPU, ...
- Examples:
  - How does ACP traffic affect cache miss rate?
  - How balanced are the busses?
  - How does changing mem access priority affect throughput?
**Evaluate Performance - Traffic Generation**

**Generate Traffic Patterns**
- Pre-defined bitstream
  - Configurable to emulate traffic patterns on multiple ports
- Simultaneous CPU loading
  - Configurable app types
- Allows for pre-porting eval

<table>
<thead>
<tr>
<th>Port Location</th>
<th>Template Id</th>
<th>Operation</th>
<th>Address Start</th>
<th>Address Next</th>
<th>Beats/txns</th>
<th>Trxn interval</th>
<th>Est. Throughput</th>
</tr>
</thead>
<tbody>
<tr>
<td>atg_acp</td>
<td>&lt;None&gt;</td>
<td>RD</td>
<td>ddr0</td>
<td>increment</td>
<td>16</td>
<td>34</td>
<td>376</td>
</tr>
<tr>
<td>atg_acp</td>
<td>&lt;None&gt;</td>
<td>WR</td>
<td>ddr1</td>
<td>increment</td>
<td>16</td>
<td>34</td>
<td>376</td>
</tr>
<tr>
<td>atg_hp0</td>
<td>&lt;Custom&gt;</td>
<td>RD</td>
<td>ddr2</td>
<td>increment</td>
<td>16</td>
<td>34</td>
<td>376</td>
</tr>
<tr>
<td>atg_hp1</td>
<td>&lt;Custom&gt;</td>
<td>WR</td>
<td>ddr3</td>
<td>increment</td>
<td>16</td>
<td>34</td>
<td>376</td>
</tr>
</tbody>
</table>
Event Trace to Dissect Timing Issues

Common Timeline
- Software Events
  - OS events (sys calls, locks, ...)
  - User events
- Hardware Events
  - Buss transactions, PL events
- Low overhead
SDSoC: FPGA Development through Software
FPGA Productivity with Technology Advancement

- CPU
- GPU
- ARM SoCs & DSPs
- Zynq SoC & MPSoC
- Zynq SoC & MPSoC

Performance / Watt & ‘Any to Any’ Connectivity

Ease of Development

VIVADO HLS

VIVADO

SDSoC Environment
Typical Zynq Development Flow

```c
APP () {
    funcA();
    funcB();
    funcC();
}
```

HW-SW partition?

- `funcA`
- `funcB, funcC`

HW-SW Connectivity?

- Datamover
  - PS-PL interfaces
  - SW drivers

- `funcA`
- `funcB, funcC`

Explore optimal architecture

Processing System (PS)

Programmable Logic (PL)
Before SDSoc:

Need to modify multiple levels of design entry
After SDSoC:

- Remove the manual design of SW drivers and HW connectivity
After SDSoc:

- Remove the manual design of SW drivers and HW connectivity
- Use the C/C++ end application as the input calling the user algorithm IPs as function calls
- Partition set of functions to Programmable Logic by a single click
After SDSoC: Automatic System Generation

C/C++

Select functions for PL

Met Req?

SDSoC

func1(); <-> SW
func2(); <-> HW
func3(); <-> HW

C/C++ to System in hours, days

Application

Driver

Datamover

PS-PL interface

IP

PL

PS
Example 1: Matrix Multiply + Add

```c
main(){
    malloc(A,B,C);
    mmult(A,B,D);
    madd(C,D,E);
    printf(E);
}
```

```
madd(inA,inB,out){
}
```

```
mmult(inA,inB,out){
}
```

SDSoC™ Environment

PS

Application

Driver

AXI Bus

Generated

PL

datamovers

© Copyright 2016 Xilinx
Example 2: 1080p60 Stereo Vision

```c
main()
{
  histEqual(A);
  histEqual(B);
  ractify(A, B, C);
  stereoBM(C, D);
  overlay(D, out);
  display(out);
}
```

Image processing on the video I/Os via DDR3 memory
How to Call Accelerators - Programming Paradigms

Explicit Message Passing APIs
- Generic API to transfer data (send/receive, set/get)
- Tasks written in C/C++ (SW) and/or VHDL/Verilog (HW)
- Mental model: Threads communicating with each other

Function call paradigm
- Standard function call paradigm
  - Synchronous or asynchronous
  - Mental model: Call an accelerator that returns result

Enqueue work items (OpenCL)
- Compile OpenCL host and kernels
- Kernels compiled to CPU/Neon or FPGA
- Mental model: Enqueue tasks to next available exec unit

High level modeling
- MathWorks - MATLAB/Simulink
- National Instruments – LabView

No “right” way of doing this – Depends on application
Summary

- **Heterogeneous systems are here to stay**
  - And they will be increasingly complex

- **Developing for heterogeneous systems is hard**
  - Each component might have its own language and operating environment
  - Parallel programming is hard to get right

- **New standards, tools, frameworks and APIs are here to help**
  - Hiding the complexity and unifying the environments

- **Don’t get stuck in old ways**
  - Embedded developers are conservative
  - Never a good time to try new methodologies
  - “Boiling frog” syndrome…