Simulation Based Analysis and Debug of Heterogeneous Platforms

Design Automation Conference, Session 60
4 June 2014

Simon Davidmann, Imperas
Agenda

- Programming on heterogeneous platforms
- Hardware-based software development, debug and test
- Software simulation – using virtual platforms
- Case study 1: Interprocessor communications with Freescale Vybrid-Kinetis platform
- Case study 2: OS porting, bring up and verification on Altera Cyclone V SoC FPGA
- Summary
Agenda

- Programming on heterogeneous platforms
- Hardware-based software development, debug and test
- Software simulation – using virtual platforms
- Case study 1: Interprocessor communications with Freescale Vybrid-Kinetis platform
- Case study 2: OS porting, bring up and verification on Altera Cyclone V SoC FPGA
- Summary
# Programming Heterogeneous Platforms

<table>
<thead>
<tr>
<th>Symmetric MultiProcessing (SMP) Architecture</th>
<th>Asymmetric MultiProcessing (AMP) Architecture</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
# Programming Heterogeneous Platforms

<table>
<thead>
<tr>
<th></th>
<th>Symmetric MultiProcessing (SMP) Architecture</th>
<th>Asymmetric MultiProcessing (AMP) Architecture</th>
</tr>
</thead>
<tbody>
<tr>
<td>Homogeneous Processor Resources</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Heterogeneous Processor Resources</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
## Programming Heterogeneous Platforms

<table>
<thead>
<tr>
<th></th>
<th>Symmetric MultiProcessing (SMP) Architecture</th>
<th>Asymmetric MultiProcessing (AMP) Architecture</th>
</tr>
</thead>
<tbody>
<tr>
<td>Homogeneous Processor Resources</td>
<td></td>
<td>Heterogeneous system</td>
</tr>
<tr>
<td>Heterogeneous Processor Resources</td>
<td></td>
<td>Heterogeneous system</td>
</tr>
</tbody>
</table>

### Diagram:
- **Mixer**
  - **RGB Out**
  - **YUV Out**
  - **Shared Memory**
  - **ARM**
## Programming Heterogeneous Platforms

<table>
<thead>
<tr>
<th>Homogeneous Processor Resources</th>
<th>Symmetric MultiProcessing (SMP) Architecture</th>
<th>Asymmetric MultiProcessing (AMP) Architecture</th>
</tr>
</thead>
<tbody>
<tr>
<td>Not heterogeneous, but often a subsystem of a heterogeneous system</td>
<td>Heterogeneous system</td>
<td></td>
</tr>
<tr>
<td>Heterogeneous Processor Resources</td>
<td>N/A (big.LITTLE?)</td>
<td>Heterogeneous system</td>
</tr>
</tbody>
</table>

- **Symmetric MultiProcessing (SMP) Architecture**: Not heterogeneous, but often a subsystem of a heterogeneous system.
- **Asymmetric MultiProcessing (AMP) Architecture**: Heterogeneous system.

### Diagram

- **Mixer**
  - RGB Out
  - YUV Out
  - Shared Memory

**Note**: The diagram shows a block diagram of a system with a mixer, RGB output, YUV output, and shared memory. The diagram indicates the flow of data between these components.
# Programming Heterogeneous Platforms

<table>
<thead>
<tr>
<th></th>
<th>Symmetric MultiProcessing (SMP) Architecture</th>
<th>Asymmetric MultiProcessing (AMP) Architecture</th>
</tr>
</thead>
<tbody>
<tr>
<td>Homogeneous Processor Resources</td>
<td>Not heterogeneous, but often a subsystem of a heterogeneous system</td>
<td>Heterogeneous system</td>
</tr>
<tr>
<td>Heterogeneous Processor Resources</td>
<td>N/A (big.LITTLE?)</td>
<td>Heterogeneous system</td>
</tr>
</tbody>
</table>

- Programming languages, tools and paradigms are broadly discussed
  - OpenCL
  - HSA/HSAIL
  - C/C++ …
  - Brute force!

![Diagram of ARM and Mixers with Shared Memory and RGB/YUV Outputs]
Example Heterogeneous Systems

- **SMP cores**
- **AMP cores**
- **Accelerators**
- **Mobile**
- **Server**
How to Analyze and Debug Heterogeneous Platforms?

- Heterogeneous systems have unique issues
  - Multiple processors
  - Different multiple processors
  - Multiple ISAs
  - Multiple operating systems
  - Hypervisors
  - Interprocessor communications (coherent and non-coherent)
  - Memory issues
  - Race/deadlock/stall conditions
  - …
 Agenda

- Programming on heterogeneous platforms
- Limitations of hardware-based software development, debug and test
- Software simulation (virtual platform) advantages for operating system porting, bring up and verification
- Case study 1: Interprocessor communications with Freescale Vybrid-Kinetis platform
- Case study 2: OS porting, bring up and verification on Altera Cyclone V SoC FPGA
- Summary
Limitations of Hardware-Based Software Development

- Traditional breadboard / emulation based testing
  - Limited physical system availability
  - Limited external test access (controllability)
  - Limited internal observability
  - Typically 6 months or more until available to team
- To get around these limitations, software is modified
  - printf added to code
  - Debug versions of OS kernels
  - Code is instrumented for specific analytical tools, e.g. code coverage, profiling
- Modified software may not have the same behavior as clean source code
Agenda

- Programming on heterogeneous platforms
- Limitations of hardware-based software development, debug and test
- Software simulation (virtual platform) advantages for operating system porting, bring up and verification
- Case study 1: Interprocessor communications with Freescale Vybrid-Kinetis platform
- Case study 2: OS porting, bring up and verification on Altera Cyclone V SoC FPGA
- Summary
Advantages of Virtual Platform Based Software Development

- Earlier system availability
- Full controllability of platform both from external ports and internal nodes
- Full visibility into platform
- Performance can be faster than real time
- Easy to replicate platform and test environment to support regression testing on compute farms
Virtual Platforms (Software Simulation)

Software is executed unchanged, such that the software does not know that it is not executing on the hardware.

- The virtual platform is a set of models that reflects the hardware on which the software will execute:
  - Subset / subsystem of a single device
  - Processor chip
  - Board
  - System

- Models are typically written in C or SystemC
- Models for individual components – interrupt controller, UART, ethernet, … – are connected just like in the hardware
- Peripheral components can be connected to the real world by using the host workstation resources: keyboard, mouse, screen, ethernet, USB, …

The diagram shows a virtual platform consisting of various components such as MIPS P5600, UART (TTY2), Memory (RAM), Malta FPGA, and other peripheral components. The virtual platform is connected to the real world through resources like keyboard, mouse, screen, ethernet, USB, and others.
Virtual Platforms Simulate the Software Running on the Hardware

Application Software & Operating System

Hardware

Peripheral
Memory
CPU

Virtual Platform simulation engine

Peripheral
Memory
OVP CPU
OVP CPU

Application Software & Operating System

results(VP) = results(HW)
Software Analysis Tools on HW Platforms Have Validity Questions

Application Software & Operating System

Hardware
- Peripheral
- Memory
- CPU

$\text{results(HW+ instrumentation)} = \text{results(HW)}$
Software Analysis Tools Using Binary Interception Techniques are Non-Intrusive

Application Software & Operating System

results(VP + instrumentation) = results(HW)
And Virtual Platform simulators can be very fast

### Example speed of Imperas simulation models

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Altera Nios II</th>
<th>ARM32</th>
<th>Imagination MIPS32</th>
<th>Xilinx MicroBlaze</th>
<th>ARM AArch64</th>
<th>Imagination MIPS64</th>
<th>PowerPC</th>
<th>Renesas v850</th>
<th>Synopsys ARC</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Simulated Instructions</td>
<td>Run time</td>
<td>Simulated MIPS</td>
<td>Simulated Instructions</td>
<td>Run time</td>
<td>Simulated MIPS</td>
<td>Simulated Instructions</td>
<td>Run time</td>
<td>Simulated MIPS</td>
</tr>
<tr>
<td>Linpack</td>
<td>3,075,857,171</td>
<td>2.52s</td>
<td>1225</td>
<td>6,105,766,856</td>
<td>4.79s</td>
<td>1277</td>
<td>9,814,621,392</td>
<td>5.31s</td>
<td>1852</td>
</tr>
<tr>
<td>Dhrystone</td>
<td>1,810,082,367</td>
<td>1.18s</td>
<td>1547</td>
<td>2,250,079,359</td>
<td>2.32s</td>
<td>974</td>
<td>1,795,086,667</td>
<td>1.27s</td>
<td>1414</td>
</tr>
<tr>
<td>Whetstone</td>
<td>5,850,887,389</td>
<td>3.28s</td>
<td>1789</td>
<td>1,185,956,501</td>
<td>1.04s</td>
<td>1140</td>
<td>1,890,420,892</td>
<td>0.93s</td>
<td>2033</td>
</tr>
<tr>
<td>peakSpeed2</td>
<td>22,000,013,458</td>
<td>3.11s</td>
<td>7097</td>
<td>22,400,009,766</td>
<td>4.67s</td>
<td>4807</td>
<td>22,800,009,853</td>
<td>4s</td>
<td>5714</td>
</tr>
</tbody>
</table>

All measurements on 3.4GHz Intel(R) i7-3770, Linux, GPGPU 20140127.0

* Hardware Floating Point Instructions
And booting OS can be fast too

- Boot Linux on ARM Cortex-A15x4 = 6 seconds on Win7 laptop
- Runs simulated Linux applications at 100s of MIPS
ARMv8 simulation using parallel host-cpu resources

- Advanced parallel synchronization algorithm for SMP, AMP and hardware accelerators
- Transparent operation to user: No model, tool, software changes
- Total performance on benchmarks recorded up to 16 Billion ins/sec
- Performance advantage 15x over nearest commercial alternative
Agenda

- Programming on heterogeneous platforms
- Limitations of hardware-based software development, debug and test
- Software simulation (virtual platform) advantages for operating system porting, bring up and verification
  - Case study 1: Interprocessor communications with Freescale Vybrid-Kinetis platform
  - Case study 2: OS porting, bring up and verification on Altera Cyclone V SoC FPGA
- Summary
Case Study 1: Interprocessor Communication on Freescale Vybrid – Kinetis Virtual Platform

- Freescale MQX RTOS running on each device
- Uses MQX interprocessor communication capability
- Open Virtual Platforms (OVP, www.OVPworld.org) models
Multicore heterogeneous debug is needed

Select target context

- Single coherent debug of cpus, peripherals, homogeneous, heterogeneous, AMP, SMP
Advanced tools needed too
MQX RTOS Scheduler Analysis

Scheduler analysis helps to identify RTOS task scheduling issues

- Imperas tools are non-intrusive
- Tools have understanding of CPUs, OSs
  - CPU-aware: code coverage, profiling, function tracing, fault injection, …
  - OS-aware: task tracing, event tracing, scheduler analysis, …
- Pre-defined and user-defined tools use same API
- Software assertions can be added to the virtual platform simulation environment
Agenda

- Programming on heterogeneous platforms
- Limitations of hardware-based software development, debug and test
- Software simulation (virtual platform) advantages for operating system porting, bring up and verification
- Case study 1: Interprocessor communications with Freescale Vybrid-Kinetis platform
  - Case study 2: OS porting, bring up and verification on Altera Cyclone V SoC FPGA
- Summary
Altera Cyclone V SoC FPGA

- ARM® Cortex™-A9MPx2
- Imperas SmartLoader
- Reset Manager
- L2 Cache Controller
- SRAM
- UART0
- UART1
- System Manager
- Ethernet
- DMA
- Timer0
- Timer1
- Timer2
- Timer3
- Memory
- jtag_uart
- uart.s1
- Timer_1ms
- Ethernet MAC
- sysid
- Flash Controller

TCP/IP stack: registered
USB hash table entries: 256 (order: 0, 4096 bytes)
USB-Lite hash table entries: 256 (order: 0, 4096 bytes)
NET: Registered protocol family 1
UDP: Registered named UNIX socket transport module,
Software use cases (challenges):

1) Linux boot on single core ARM Cortex-A9
2) SMP Linux boot on dual core ARM Cortex-A9
3) RTOS boot on single core ARM Cortex-A9
4) AMP boot on dual core ARM Cortex-A9
5) Linux boot on single core Nios II
6) SMP Linux boot on dual core ARM Cortex-A9 plus Linux boot on Nios II
Cyclone V SoC FPGA Virtual Platform

- Top level virtual platform built using Open Virtual Platforms (OVP, www.OVPworld.org) ICM API
- ARM Cortex-A9MPx2 and Altera Nios II processor core models from the OVP Library
- Peripheral models
  - Some models available in the OVP Library
  - Remaining models of peripheral components developed using OVP APIs
- OVP APIs written for C language
- Simulation engine: Imperas M*SDK

All OVP processor and peripheral models include both native OVP and native SystemC/TLM2 interfaces, so all the following results could have been achieved using the OSCI SystemC simulator plus Imperas M*SDK product
  - Peripheral models could have been written in SystemC
  - M*SDK tools require OVP processor core models for ToolMorphing capability
1a) Linux Boot on Single Core ARM Cortex-A9

- Use Linux from Altera: Altera-3.4
- Use default configurations
- Use default device trees
  - Comment out a few peripherals not yet modeled
- Bug found in Linux kernel preemptive scheduling
  - Running multiple applications under Linux part of standard Imperas bring up testing
  - Linux boots and runs, but does not switch tasks properly
  - Not observed in previous virtual platform (different virtual platform vendor) using much slower model of ARM Cortex-A9MPx2
    - Could not run multiple applications for long enough simulation to observe the bug

- Approximately 2 man weeks effort to build virtual platform able to boot Linux
- Virtual platform boots Linux in under 5 sec on standard PC, Windows or Linux
1b) OS-Aware Tools Used to Find the Bug

- Use OS tracing [task, execve, schedule, context, …] to trace at the OS level, not instruction level
  - Higher level of abstraction makes debug easier: ~700,000,000 to boot Linux, however, only ~700 tasks
- OS-aware tools debug in hours, once the bug was observed
- Simulation overhead due to OS-aware tools < 10%
2) SMP Linux Boot on Dual Core ARM Cortex-A9

- Use Linux from Altera: Altera-3.6
- Use default configurations
- Use default device trees
  - Comment out a few peripherals not yet modeled
- No problems in SMP Linux bring up on virtual platform
3a) Micrium µCOS-II Boot on Single Core ARM Cortex-A9

- Use Altera µCOS-II release
- Bugs found and fixed in GIC register accesses using OS-aware tools
  - Access ICDICER 1 to 8 when only 0 to 7 exist
  - Access ICDIPTR 08 to 63 when only 00 to 55 exist
- Typically < 1 week effort to add support for new RTOS
- RTOS OS-aware tools include event scheduler viewing as waveform
Non-intrusive (no modification of OS source) trace of
- process creation
- context switch
- process deletion
Captures communications between processes

Supported OS include Linux, FreeRTOS, Nucleus, μC/OS
- < 1 week to support new RTOS
- View in waveform viewer
4a) AMP boot on Dual Core ARM Cortex-A9

- Linux booting on first core, µC/OS-II on second core
- Bug found in Linux accesses of GIC registers
- Virtual platform debug took 2 days versus 2 weeks on hardware platform (5x improvement)
- Also need to ensure that different operating systems do not access forbidden memory segments
4b) Custom Memory Access Monitor Accelerates AMP Platform Debug

- Memory access monitor is just C code, less than 350 lines, loaded into simulation environment
- When simulation is run, monitor produces warning if memory access rules are violated

```c
// Define watch areas for memory and peripherals defined in the platform
memWatchT amcWatch[] = {
    // name                        watchLow    watchHigh   allowedCPUs
    { "Linux memory",            0,          0x2fffffff, LINUX_CPU   },
    { "uCOS memory",             0x30000000, 0x31fffffff, UCOSII_CPU  },
    { "gmac0",                   0xff700000, 0xff701fff, LINUX_CPU   },
    { "gmac1",                   0xff701000, 0xff702fff, LINUX_CPU   },
    { "eMAC0 dma",               0xff703000, 0xff704fff, LINUX_CPU   },
    { "eMAC1 dma",               0xffc02000, 0xffc03fff, LINUX_CPU   },
    { "uart0",                   0xffc03000, 0xffc04fff, UCOSII_CPU  },
    { "CLKMGR",                  0xffd04000, 0xffd04fff, LINUX_CPU   },
    { "RSTMGR",                  0xffd05000, 0xffd05fff, LINUX_CPU   },
    { "SYSMGR",                  0xffd06000, 0xffd06fff, LINUX_CPU   },
    { "GIC",                     0xffec0000, 0xffedfff, LINUX_CPU   },
    { "L2",                      0xff000000, 0xff00fff, LINUX_CPU   },
    { 0 } /* Marks end of list */
};
```

Warning (AMPCHK_MWV) cpu_CPU0: AMP write access violation in uart1 area. PA: 0xffc03008 VA: 0xffc03008
Warning (AMPCHK_MWV) cpu_CPU0: AMP write access violation in uart1 area. PA: 0xffc0300c VA: 0xffc0300c
Warning (AMPCHK_MWV) cpu_CPU0: AMP write access violation in uart1 area. PA: 0xffc03010 VA: 0xffc03010
Warning (AMPCHK_MRV) cpu_CPU1: AMP read access violation in Linux memory area. PA: 0x00000020 VA: 0x00000020
5) Linux on Single Core Altera Nios II

- Altera Cyclone III 3c120
- Linux booting on Nios II processor core model
- No issues with Linux boot

---

Welcome to Nios II
nico2 login: 

```
TCF: reno registered
UDP hash table entries: 256 (order: 0, 4096 bytes)
UDP-Lite hash table entries: 256 (order: 0, 4096 bytes)
NET: Registered protocol family 1
RPC: Registered named UNIX socket transport protocol
RPC: Registered udp transport protocol
RPC: Registered tcp transport protocol
RPC: Registered tcp over IPv4 transport protocol
jiffS2: version 2.2, (NVIDIA) @ 2001-2006 Red Hat, Inc.
imagen has been set to 238
Block layer SCSI generic (bsg) driver version 0.4 loaded (major 254)
io scheduler noop registered
io scheduler deadline registered
io scheduler cfq registered (default)
ttysLO at MMIO 0x8000ea80 (irq = 10) is a Altera UART
ttyJO at MMIO 0x8000f50 (irq = 1) is a Altera JTAG UART
iogt: module loaded
noaccess: F5/2 mouse device common For all mice
TCF: cubic registered
NET: Registered protocol family 17
turn off boot console early)
```
6) Heterogeneous AMP Platform

Altera Cyclone V Cortex A9MPx2 (SMP Linux) and Nios II (Linux)
Summary

- Heterogeneous platforms require not only new programming tools but also new debug and analysis tools.
- Hardware-based software testing has limitations in controllability and observability.
- Instruction accurate software simulation – virtual platforms – has the required controllability and observability.
- Use of virtual platform based tools can provide both higher quality and reduced schedules for heterogeneous systems.
- Results were shown for heterogeneous ARM-ARM on Freescale Vybrid and Kinetis, and for ARM-Nios II AMP system on Altera Cyclone V SoC FPGA.