Virtual Platform Software Simulation for Enhanced Multi-core Software Verification

Simon Davidmann (simond@imperas.com)
Company: Imperas Software Ltd, 17 March 2014
Event: TVS – Software Testing
Location: UWE Conference Centre, Bristol
Agenda

- Background – the changing needs of embedded product development
- “Traditional” Embedded Software development solutions
- Working with Multi-core hardware and software
  - The next generation of Embedded Software development systems
- Example – using assertions on software
  - shared memory monitor
- Summary
The changing hardware

- SMP cores
- AMP cores

- Multi-core [is going to be in] [is in] everything
- AMP, SMP, homogeneous, heterogeneous
The changing software

Layer and Layers of software
The changing SW Verification Requirements

- **Hardware Dependent Software (HDS)**
  - Most complex foundation layer
    - Drivers, hypervisors, assembly libraries, Operating System
  - Buried problems often appear elsewhere in a system, leading to misdirected analysis
  - Ripe for corner case type issues
  - Post development bugs hardest to fix
  - Testing needs to be platform centric not application centric

- **Modern SoC verification is complex**
  - SMP/AMP Multicore Interaction
  - Shared memory & devices
  - Extensive accelerators, peripherals
  - Complex SW/HW interaction (e.g. security)
  -Externally authored, complex libraries
Current Solutions For Early Embedded SW Development

Traditional Breadboard
- Limited system availability
- Limited external test access
- Limited internal visibility
- Late in arrival

Emulation & Cycle Accurate Models
- Provides reasonable verification capability but 1000x too slow for effective HDS verification
- Hard to get started
Agenda

- Background – the changing needs of embedded product development
- “Traditional” Embedded Software development solutions

- Working with Multi-core hardware and software
  - The next generation of Embedded Software development systems

- Example – using assertions on software
  - shared memory monitor

- Summary
## Learning from Hardware Verification for Multi-core

- In the beginning it was all about prototype exploration
- Then it was simulators and HDLs
- Then came dynamic verification
- Then came formal

For software development - what can we learn - some examples:

1. Simulation is necessary, but is not sufficient
2. An assertion based methodology confirms correct operation
3. Functional coverage allows measurement and focuses test development
4. … (yes there is much more)
Virtual Platforms: Transforming Engineering Efficiency

SW vs HW development platforms
- Often available earlier for engineering
- Provide more powerful tooling
- Easier to replicate for extended testing
- Real time or faster performance

“Best-in-class manufacturers that make extensive use of simulation early in the design process hit revenue, cost, launch date, and quality targets for 86% or more of their products. Best-in-class manufacturers of the most complex products get to market 158 days earlier with $1.9 million lower costs than all other manufacturers.”

Simulation-Driven Design, Aberdeen Group Study
And Virtual Platform simulators can be very fast

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Simulated Instructions</th>
<th>Run time (s)</th>
<th>Simulated MIPS</th>
<th>Run time (s)</th>
<th>Simulated MIPS</th>
<th>Run time (s)</th>
<th>Simulated MIPS</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Altera Nios II</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Linpack</td>
<td>3,075,087,171</td>
<td>2.52</td>
<td>1225</td>
<td>4.79</td>
<td>1277</td>
<td>5.31</td>
<td>1852</td>
</tr>
<tr>
<td>Dhrystone</td>
<td>1,810,082,387</td>
<td>1.18</td>
<td>1547</td>
<td>2.32</td>
<td>974</td>
<td>1.27</td>
<td>1414</td>
</tr>
<tr>
<td>Whetstone</td>
<td>5,850,987,389</td>
<td>3.28</td>
<td>1789</td>
<td>1.04</td>
<td>1140</td>
<td>0.93</td>
<td>2033</td>
</tr>
<tr>
<td>peakSpeed2</td>
<td>22,000,013,468</td>
<td>3.11</td>
<td>7097</td>
<td>4.67</td>
<td>4807</td>
<td>4s</td>
<td>5714</td>
</tr>
<tr>
<td><strong>Xilinx MicroBlaze</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Linpack</td>
<td>6,386,275,169</td>
<td>3.77</td>
<td>1699</td>
<td>1.01</td>
<td>594*</td>
<td>0.83</td>
<td>1901</td>
</tr>
<tr>
<td>Dhrystone</td>
<td>3,770,115,740</td>
<td>2.61</td>
<td>1450</td>
<td>2.79</td>
<td>1086</td>
<td>1.23</td>
<td>1293</td>
</tr>
<tr>
<td>Whetstone</td>
<td>27,108,632,665</td>
<td>13.23</td>
<td>2054</td>
<td>0.64</td>
<td>759*</td>
<td>0.99</td>
<td>2156</td>
</tr>
<tr>
<td>peakSpeed2</td>
<td>22,000,023,433</td>
<td>5.76</td>
<td>3826</td>
<td>3.73</td>
<td>3011</td>
<td>4.23</td>
<td>4052</td>
</tr>
<tr>
<td><strong>ARM AArch64</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Linpack</td>
<td>1,364,586,113</td>
<td>2.66</td>
<td>1076</td>
<td>4.76</td>
<td>1051</td>
<td>3.67</td>
<td>1143</td>
</tr>
<tr>
<td>Dhrystone</td>
<td>2,206,538,239</td>
<td>1.76</td>
<td>1260</td>
<td>4.01</td>
<td>1603</td>
<td>2.76</td>
<td>1148</td>
</tr>
<tr>
<td>Whetstone</td>
<td>6,424,655,766</td>
<td>3.97</td>
<td>1622</td>
<td>7.41</td>
<td>1393</td>
<td>4.4</td>
<td>1796</td>
</tr>
<tr>
<td>peakSpeed2</td>
<td>22,400,002,937</td>
<td>5.66</td>
<td>4007</td>
<td>3.53</td>
<td>6364</td>
<td>4.06</td>
<td>5446</td>
</tr>
</tbody>
</table>

Example speed of Imperas simulation models
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
Software Quality is Directly Proportional to Test Speed

Less Time
Greater Quality
More Bugs Found
More Tests Run
Faster Tests
Virtual Platforms: The Right Performance to Capability Trade-off

Virtual Platforms (simulators) with Instruction Accurate (IA) models provide:
- Pre-prototype verification
- Effective verification access
- Reasonable execution performance

However, Virtual Platforms require a simulation foundation to be effective
- Standardized modeling technology
- Services for verification tools
- Tool firewall for execution integrity
- Make use of host parallel resources for maximum performance
OVP Standardized Modeling Infrastructure

Open Virtual Platforms™ (OVP™) standardized set of Modeling APIs for platforms, cpu models and behavioral peripheral models
Simulator Architecture to provide services for Verification

- Leveraging JIT Code Morphing simulation algorithms for highest possible performance
- Modeling APIs allow processor/platform functionality to be described efficiently while maintaining easy modeling environment
- Imperas technology allows verification and debug tool code to be combined with model and software execution efficiently and unobtrusively
Traditional Debug (1D)

- Debugging application code running on simulated embedded processor

- Trace
  - e.g.: instructions, source lines, register changes

- GDB-like debugger

- Examining registers, variables, source...

- Single step, breakpoints, …

- GUI
Spatial & Temporal Debug (2D)

Spatial

- For AMP/SMP – examine applications on multiple cores across the chip
- Debugging peripheral/behavioral models in context of software running on the embedded cores
  - Programmers view, or model source

Temporal

- Considering the sequences of events over time
  - Sequential assertions, breakpoints
  - Using conditioning events to prime breakpoints
    - e.g.: break on next ISR after character input to UART
Layer-Aware Debug (3D)

Layered verification matches layered software architecture
Simulator must allow focus and stratification

OS/CPU-Aware Focused Debug
Commands analyze layer operation while excluding irrelevant detail
(easier to view 1000 tasks operations than 1 Billion instructions trace)

Layer-aware Stratified Analysis
Connecting commands through different layers for activity analysis

TRC (SCHD) 242131778: 'mipsle1_TCO': scheduler switched
("khelper")
TRC (TASK) 242137813: 'mipsle1_TCO': do_execve called f
TRC (EXEC) 242137813: 'mipsle1_TCO': do_execve called f
filename=/sbin/hotplug with:
TRC (EXEC) 242137813: 'mipsle1_TCO':   argv virt=0x80
TRC (EXEC) 242137813: 'mipsle1_TCO':   argv virt=0x80
TRC (EXEC) 242137813: 'mipsle1_TCO':   envp virt=0x80
TRC (EXEC) 242137813: 'mipsle1_TCO':   envp virt=0x80
"PATH=/sbin:/bin:/usr/sbin:/usr/bin"
Rich set of commands to operate at all layers of abstraction
Extensible Architecture Enables Platform / OS Specific Tooling

Extended Platform Capabilities
- OS Capabilities
- CPU Capabilities

Application Software & Operating System

Virtual Platform
- Memory
- Peripherals

CPU Assistant

OS Assistant

Extended / Standard Tools

Output Data (e.g. Schedule, Coverage)

Multi-processor Multi-core 3Debug™ Debugger
## Agenda

- Background – the changing needs of embedded product development
- “Traditional” Embedded Software development solutions
- Working with Multi-core hardware and software
  - The next generation of Embedded Software development systems
- Example – using assertions on software
  - shared memory monitor
- Summary
Example 1 - Assertions
Altera Cyclone V Cortex A9MPx2 (AMP Linux/Micrium, SMP Linux) and Nios II (Linux)
**Verification challenges...**

OS Porting, Bring Up and Verification on Altera Cyclone V SoC FPGA

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
Memory access monitor is just C code, 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[] = {
  { "Linux memory", 0, 0x2fffffff, LINUX_CPU },
  { "uCOS memory", 0x30000000, 0x31fffffff, UCOSII_CPU },
  { "gmac0", 0xff700000, 0xff700fff, LINUX_CPU },
  { "emac0dma", 0xff701000, 0xff701fff, LINUX_CPU },
  { "gmac1", 0xff702000, 0xff702fff, LINUX_CPU },
  { "emac1dma", 0xff703000, 0xff703fff, LINUX_CPU },
  { "uart0", 0xffc02000, 0xffc02fff, LINUX_CPU },
  { "uart1", 0xffc03000, 0xffc03fff, UCOSI_CPU },
  { "CLKMGR", 0xffd04000, 0xffd04fff, LINUX_CPU },
  { "RSTMGR", 0xffd05000, 0xffd05fff, LINUX_CPU },
  { "SYSTMGR", 0xffd08000, 0xffd08fff, LINUX_CPU },
  { "GIC", 0xfffc0000, 0xffffedfff, LINUX_CPU },
  { "L2", 0xffffef000, 0xffffffffff, LINUX_CPU },
};
```

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
## Summary of verification example

1) Linux boot on single core ARM Cortex-A9
   - Bug found in Linux kernel preemptive scheduling
     - 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

2) SMP Linux boot on dual core ARM Cortex-A9
   - OK – no problems found

3) RTOS boot on single core ARM Cortex-A9
   - Bugs found and fixed in GIC register accesses using OS-aware tools

4) AMP boot on dual core ARM Cortex-A9
   - Bug found in Linux accesses of GIC registers
   - Bugs found in RTOS access of Linux’s reserved memory

5) Linux boot on single core Nios II
   - No problems found

6) SMP Linux boot on dual core ARM Cortex-A9 plus Linux boot on Nios II
   - No problems found
## Summary

- Simulation is necessary but not sufficient
  - Fast simulation finds more bugs
  - Making use of multi-core host is even better
- Trace, temporal, and multi-core debug are essential for AMP/SMP systems
- ‘Layer-aware’ analysis makes debug manageable
  - Allows focus at different levels of abstraction
- Ability to extend functionality and write own tools are the key to providing efficient development environments
Conclusions

- It is inevitable that simulation will form the basis of the next generation of embedded software development methodology.
- Ensure your chosen simulator is fast, has a standardized modeling capability, and has the ability to include integrated advanced tools.
- 2D and 3D verification, analysis and debug tools are essential for multi-core designs.
- To find the most complex bugs and ensure product quality an advanced verification approach is needed using layered, customizable tools.
Thank you

♫ For more modeling/model information
♫ www.OVPworld.org

♫ For technology/product information
♫ www.imperas.com

♫ For Simon: simond@imperas.com

♫ Questions?