An Approach to Verification of Many-Core Systems Using the Virtual Platform

Victoria Mitchell
Altera Corporation
Previously, on this program...

Extending the life of the Virtual Platform:
- Earlier Verification
- Architectural Decisions
- Design Verification (visibility)
- DFX Methodology

... not just early enablement of software, but true simultaneous engineering
Why Many-Core?

1. Parallelization of tasks
2. Using cores as repeatable templates of custom logic
   - Modify functionality over time
   - Faster to design
   - More automatic and reproducible
   - Ultimately more configurable
Why Extend The Virtual Platform?

1. Cost effective to leverage investment
2. Develop once, use in many applications
3. Improves overall quality of HW
4. Improves overall quality of system

Enabled by unique capability of the VP
- Instrumentation of platform: models, peripherals
- Intercepting simulation with hosted functions
- Non-intrusive
- Verify the full system
Tools Used: Processor Simulation and Interception

- Translates instructions to host native
- Dynamically builds translation lookup
- Peripheral models execute in quantum measure of time
How Binary Intercepts Work

- Dynamic loadable modules
- APIs are registered to events
- Examples of events:
  - simulation construction
  - model enumeration
  - before or after an instruction morph
  - after 1-N instructions
  - when address is executed
  - when data address range is accessed
  - programmers view events
- May be opaque or transparent
- What you can do with intercept APIs
  - inspect memory
  - drop into debugger of simulation, or code
  - alter translation
  - change processor state
  - evoke other APIs
  - add/remove other API callbacks
### Imperas M*SDK and VPA API

#### Use cases include
- Drivers
- Firmware
- Assembly libraries
- OS porting and bring up
- Hypervisors

#### Tool features
- Multiprocessor, multicore, multithread, multi-everything
- Non-intrusive
- Low overhead (high performance sim)
- User extendable

---

**Operating System**

**Bare Metal Apps & Middleware**

**Platform (e.g. Drivers)**

<table>
<thead>
<tr>
<th>Processor</th>
<th>Simulator</th>
</tr>
</thead>
<tbody>
<tr>
<td>Trace coprocessor registers</td>
<td>Break on messages</td>
</tr>
<tr>
<td>Trace TLB trace exceptions</td>
<td>TCL callbacks</td>
</tr>
<tr>
<td>Trace modes</td>
<td>Full GDB command set</td>
</tr>
<tr>
<td>Trace service calls</td>
<td>Break on line</td>
</tr>
<tr>
<td>Trace hypervisor calls</td>
<td>Break on function call</td>
</tr>
<tr>
<td>Trace secure monitor calls</td>
<td>Elf introspection</td>
</tr>
<tr>
<td>Trace MT/MP extensions</td>
<td>Unlimited HW breakpoints</td>
</tr>
<tr>
<td>Trace system calls</td>
<td>Memory region watchpoints</td>
</tr>
<tr>
<td>Trace timer</td>
<td>Trace peripheral access</td>
</tr>
<tr>
<td>Trace cache instructions</td>
<td>Memory coverage</td>
</tr>
<tr>
<td>Trace SIMD extensions</td>
<td>Shared memory checks</td>
</tr>
<tr>
<td>Trace instruction</td>
<td>Bus connectivity view</td>
</tr>
<tr>
<td>Trace register change</td>
<td>Peripheral register view</td>
</tr>
</tbody>
</table>

**Tool features**

- Multiprocessor, multicore, multithread, multi-everything
- Non-intrusive
- Low overhead (high performance sim)
- User extendable
Our Platform Setup

- Matrix network of NIOS-II soft cores
  - 1 "master"
  - 2 - N cores
  - many topologies

- NIOS-II OVP model for nodes

- Node functionality
  - initialization
  - address negotiation
  - data packet handling

- Platform construction/assembly
  - specify # NIOS and topology
Platform With Direct Verification

- Platform runtime w/ code embedded in NIOS-II SW
  - Sendmsg() function
  - Acknowledge() function
- Uses printf()
- Compile-time switches to enable
- Changes execution path and size of the code

Code Here
Platform With Intercept Library

- Same platform, but with “production-ready” NIOS firmware
- Intercept sendmsg() and acknowledge()
- Registered callback at memory access

Code Here
Positive Feedback

- Use of intercepts eliminate need to change NIOS-II microcodes
- Validation engineers hook into intercept functions where they would normally write code for a directed test
- Intercept functions have simple CLI that is scriptable onto test bench
- AV is confirmed for addressing, topology, transactions
- DV validates on the system level
- eSW focuses on production code
Improvements Realized

**Speed of Design**
- FW starts with test-bench setup
- Closed-loop architecture verification

**Test Coverage**
- RTL and FW in sync
- 100% production microcode coverage

© 2014 Altera Corporation—Public
Still Work to be Done

1. Separate instances of the intercept library per processor
   - Memory impact
   - Duplication and sync

2. Execution speed vs number of cores

3. Enhance instrumentation to include FW profiling and performance
Conclusions

Extending the Software Virtual Platform:

- Doesn't replace timing analysis or characterization of the design, but …
  - Improvements are well worth the negligible extra effort

- We plan to continue to use thru product development lifecycle of complex, many-core systems
Thank You