Verifying Oracle’s SPARC processors with ACL2

Greg Grohoski
VP, Hardware Development
Oracle Microelectronics

May 23, 2017
Outline

• SPARC refresher

• Processor verification challenges

• ACL2 experience

• Q&A
**SPARC @ Oracle**

<table>
<thead>
<tr>
<th>Year</th>
<th>Model</th>
<th>2nd Gen Cores</th>
<th>3rd Gen Cores</th>
<th>4th Gen Cores</th>
</tr>
</thead>
<tbody>
<tr>
<td>2010</td>
<td>T3</td>
<td>16 x 2nd Gen</td>
<td>8 x 3rd Gen</td>
<td>6 x 4th Gen</td>
</tr>
<tr>
<td></td>
<td></td>
<td>cores</td>
<td>Cores</td>
<td>Cores</td>
</tr>
<tr>
<td></td>
<td></td>
<td>6MB L2 Cache</td>
<td>4MB L3 Cache</td>
<td>64MB L3 Cache</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1.7 GHz</td>
<td>3.0 GHz</td>
<td>4.1 GHz</td>
</tr>
<tr>
<td>2011</td>
<td>T4</td>
<td>16 x 3rd Gen</td>
<td>16 x 3rd Gen</td>
<td>32 x 4th Gen</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Cores</td>
<td>Cores</td>
<td>Cores</td>
</tr>
<tr>
<td></td>
<td></td>
<td>8MB L3 Cache</td>
<td>4MB L3 Cache</td>
<td>16MB L3 cache</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3.0 GHz</td>
<td>3.6 GHz</td>
<td>4.2 GHz</td>
</tr>
<tr>
<td>2013</td>
<td>T5</td>
<td>6 x 3rd Gen</td>
<td>12 x 3rd Gen</td>
<td>8 x 4th Gen</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Cores</td>
<td>Cores</td>
<td>Cores</td>
</tr>
<tr>
<td></td>
<td></td>
<td>6MB L3 Cache</td>
<td>48MB L3 Cache</td>
<td>64MB L3 Cache</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3.6 GHz</td>
<td>3.6 GHz</td>
<td>4.1 GHz</td>
</tr>
<tr>
<td>2013</td>
<td>M5</td>
<td>16 x 3rd Gen</td>
<td>32 x 4th Gen</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Cores</td>
<td>Cores</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>8MB L3 Cache</td>
<td>64MB L3 Cache</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>3.6 GHz</td>
<td>4.1 GHz</td>
<td></td>
</tr>
<tr>
<td>2015</td>
<td>M6</td>
<td>6 x 3rd Gen</td>
<td>12 x 3rd Gen</td>
<td>8 x 4th Gen</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Cores</td>
<td>Cores</td>
<td>Cores</td>
</tr>
<tr>
<td></td>
<td></td>
<td>48MB L3 Cache</td>
<td>48MB L3 Cache</td>
<td>16MB L3 cache</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3.6 GHz</td>
<td>3.6 GHz</td>
<td>4.2 GHz</td>
</tr>
<tr>
<td>2016</td>
<td>M7</td>
<td>8 x 4th Gen</td>
<td>32 x 4th Gen</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Cores</td>
<td>Cores</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>16MB L3 cache</td>
<td>64MB L3 Cache</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>4.2 GHz</td>
<td>4.1 GHz</td>
<td></td>
</tr>
<tr>
<td></td>
<td>M8</td>
<td>Next up</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Copyright © 2017 Oracle and/or its affiliates. All rights reserved.
SPARC M7

• World’s fastest microprocessor
  – Highest throughput of any commercial processor
  – 20 commercial benchmark records

• 32, 8-threaded 2-issue OOO cores, 4.1 GHz, 64MB partitioned L3 cache, 160 GB/s memory BW

• Most secure
  – Silicon Secured Memory (SSM)
  – Widest cipher acceleration (DES/3DES, AES, Camellia, MD5, all SHA variants, public-key generation including ECC) – always-on encryption

• Numerous software-in-silicon features
  – Database acceleration (DAX)
M7 Processor Overview

- 32 SPARC 8-threaded C4 cores @ 4.1 GHz
- 8 partitions each with a core cluster and 8MB of L3$
- 8 Data Analytics Accelerators for query acceleration and messaging
- 8 DDR4 memory schedulers providing 160 GB/s
- A coherency subsystem for 1 to 32 socket scaling
- A coherency and IO interconnect that provides > 336GB/s of peak BW
- Extensive SW-in-Silicon feature set
Silicon Secured Memory

- SPARC M7 protects data in memory
- Hidden “color” bits added to pointers (key), and content (lock)
- Pointer color (key) must match content color or program is aborted
  - Set on *memory allocation*, changed on *memory free*
  - Protects against access off end of structure, bugs, stale pointer access, and malicious attacks
SQL in Silicon: Data Analytics Accelerator (DAX)

- Stream Processor for data parallel operations
- DSP-like pipe for efficient filtering operations, typical of first phase of any query
- Cache sparing design with more complex processing in general purpose cores
Processor development challenges

• Increasing complexity
• Constrained development cost, schedule
• Long, lengthening fab times
• High mask costs
• Processor verification tools, techniques have plateaued

→ Increased risk of late-cycle bugs
→ Geometric increase in bug cost vs. time
Example: processor core bugs vs. time beyond first TO
What to do...

- Simplify, leverage (of course...)
- Use more simulation cycles, resources
  - Not helping much – $2^{x00} \rightarrow 2^{x,000}$ control states
- Take more development time, tape-outs
  - Not feasible
- Reconfigurable logic?
  - Microcode (for some classes of bugs...)
- Embrace formal verification techniques
  - Had been using model checking, but...what about theorem proving?
Summer 2013

• Improving divide, square root performance for new core via new algorithm
  – Floating-point divider (32b and 64b)
  – Floating-point square root (32b and 64b)
  – Integer divider (signed and unsigned, 32b and 64b)

• How to verify?
  – Stand-alone testbench – impossible to cover all operand values
  – Mathematical proof – needed expertise

• Asked for help from Oracle Labs
• Labs contacted UT Austin
  – ACL2
Case study

• Floating-point division and square root verification known to be difficult
  – Project was estimated to be a 2 year effort
• We had less than 1 year to RTL freeze to find any errors
• Team started internally from scratch

• Vision for project:
  • *Apply additional formal verification techniques to increase confidence in correctness of our designs*
    – Improve pre- and post-silicon verification
Summer 2014

• Wrote 1st draft of ACL2 spec for
  • IEEE 754 Standard on Floating-Point Arithmetic
  • Integer division variants
• Verified
  • Floating-point implementations wrt significand
  • Integer division implementations
• Found improvements for SPARC core
  • Reduced look-up tables by 60%
  • Improvements based on careful error analysis
  • Simplified square-root implementation
Summer 2015

- Validated ACL2 spec of IEEE 754 Standard against 8M vectors
- Verified all 4 floating-point implementations wrt
  - Sign, exponent, and significand
  - All exceptions
- Found more improvements for future SPARC core
  - Reduced latency further for
    - fdivs, fdivd
    - fsqrts, fsqrtd
    - idiv
- Improvements were all proven correct first and then implemented
  - No extensive 24x7 testing needed
Summer 2016

• Applied ACL2 to other areas
  • Temporal properties
  • Proved absence of starvation for certain instructions
• Looked at verification of complex crypto instructions
  • Verified Montgomery algorithms implementations wrt their math specification
  • Saved 25% latency in one instruction
  • Generated test vectors to exercise special cases for these instructions
• Verified and improved fault-tolerant cache-coherency protocol (like CCIX)
  – “Bake off” between ACL2 (TP) and Murphi/PReach (MC)
Summer 2017

- Found more improvements for integer division for future SPARC core
  - Further, significant latency reduction for integer division
  - Again, optimization was first verified, then implemented
- Verified another function using Cellular Automata Shift Registers
- Applied ACL2 to prove absence of starvation for certain instructions
  - Another “bake off” between ACL2 (TP) and SVA model checking
Future directions

• Increase use of theorem-proving where possible
  – State machines?
  – Help validate inter-module protocols
• Improve tools
  – Scale (to better handle inter-module behavior)
  – Automate extraction from Verilog to ACL2
• Need to simplify/abstract expertise needed to use ACL2
• …
Contributions to the Community

- We thank the ACL2 community for all their contributions
- A healthy ACL2 community is vital
  - Oracle funds PhDs at UT Austin
  - Oracle supports the ACL2 workshop
  - Oracle contributes improvements to ACL2 libraries
  - Oracle contracts with FHI
Contributors

• Oracle
  – David Rager
  – Jo Ebergen
  – Dmitry Nadezhin
  – Austin Lee
  – Andrew Brock
  – and many others...

• UT Austin
  – Cuong Chau
  – Ben Selfridge
  – Keshav Kini
  – Warren Hunt
  – Matt Kaufmann
Hardware and Software
Engineered to Work Together