Using Formal Methods in an Organization with a Simulation-Based Mentality

IWLS 2017

Anna Slobodova
Formal Verification Team

Shilpi Goel

Rob Sumners

Anna Slobodova

Sol Swords
Centaur Technology

Founded in 1995 as a startup to lower cost of x86 processors
100 people; constant for over ten years
About Us

- Focus on low power, low cost, and high performance x86 processors
- Parts are sold as products of VIA Technologies / Zhaoxin
- 15 major designs shipped
- Customers: IBM, Dell, Samsung, HP, Lenovo
Challenges of x86 Design and Validation

- Target is x86 instruction-set architecture.
  - A very low-level of x86 ISA compatibility required.
  - x86 ISA is complicated, under-specified, & evolving.
- In early days, we were burnt making some “who would ever care about that” assumptions:
  - Undefined flags
  - Undefined A20 “interrupt” handling
  - Undefined uncacheable fetch behavior
  - Hidden processor memory areas
  - Reserved bit exceptions
  - Model-specific registers usage
- There are many versions of the implementation.
Reasons to Consider FV at Centaur

- Testing cannot cover increasing number of instructions and widening operands
Early Formal Verification Projects at Centaur

- Does a sequence of micro-operations implement the 64-bit integer divide?
  - Yes. FV found no bugs here.
- Does a new and fast FP addition implementation meet its specification?
  - FV introduced late in design cycle after thorough testing was done.
  - Corner case found --- ~1 in $2^{144}$ chance!
- Bugs found late in the design showed shortcomings of EDA vendors’ transistor-level design validation tools
  - FV team created a formal model of the transistor-level design for equivalence checking
Centaur Formal Verification Tools Today

Based on ACL2

- First FV effort at Centaur was successful
- Publicly available
- Rich ACL2 libraries
- In use/development for 25+ years
- Flexible for building point and general-purpose tools
- ACL2 developers, interns, and future employees in town!
FV Tool Flow at Centaur

- From SystemVerilog RTL and Specifications to ACL2 Proofs:
Where FV Can Help #1

Specifying and verifying data-intensive arithmetic and logic operations

- 64- to 512-bit wide integer and floating-point arithmetic and logical operations
  - Increase in number of cases due to write-masking
  - The Media Integer Unit executes over 470 MMX, SSE, and AVX instructions, not counting the various forms of each instruction.

...I have implemented the MINT unit for several projects without a single bug in silicon. I feel that formal verification has been a key factor in this happy story.

Tom Glover
Designer
Where FV Can Help #2

Verification of self-contained microcode blocks

E.g.: 64-bit integer multiply, 64-bit integer divide, SSE 4.2 string compare instructions, etc.
Where FV Can Help #3

Maintaining correctness across different design versions

● Re-running proof regressions whenever design is modified
  ○ Design is growing to cover more functionality
  ○ Design is changing in the design exploration phase
  ○ Design is being optimized due to timing/power considerations
● Checking the correctness of bug fixes

… when the bug was fixed, formal verification quickly demonstrated correctness, rather than having to directly create and analyze exhaustive test vectors.

● Fast rerun of proofs allows late design changes
Verifying memory operations

- Goal: prove MP memory operations always complete with legal results.
- Communication ring routes requests, responses, credits, etc. between endpoints (Cores, LLCs, IO, DRAM).
- Current focus: Prove all operations make progress to completion in L2+LLC+Ring.

Formal approaches to searching for deadlocks, starvation bugs, and credit leaks will be important tools… for verifying our memory hierarchy going forward.
Where FV Can Help #5

Having a formal model of the RTL design allows various kinds of analyses:

- Static checking: e.g., linting
- Functional
- Structural
- Control- and Data-Flow
- Dependency
Where FV Can Help #6

- Checking design-specific (internal) properties on demand
- Equivalence of different versions of the design
- Verification of new algorithms for instruction implementation
  - E.g., Fused Multiply-Add
- Assistance with late changes in design (ECO)
- Mechanically-generated web-based documentation of ISA and microarchitecture-level behavior and the proofs’ status
- Map post-synthesis signals to RTL design signals to interpret timing reports
- Expand test database for DV validation of floating-point instructions
Where FV Probably Can’t Help

- When specification is as complex as the design
  - Specifying and verifying the processor front-end: does a sequence of ISA-level instructions translate “correctly” to a sequence of micro-operations?

- Compatibility with Intel machines in under-specified and/or ambiguous features
  - E.g.: legacy modes, model-specific registers, CPUID-specific behavior, etc.

- When the design interface is not “clean”

- Low-level functionality (e.g., power management)

- Interplay of microcode and hardware
Questions?