Industrial Use (of ACL2) for Hardware Verification

Warren A. Hunt, Jr.

Joint work with: Anna Slobodova, Bob Boyer, Shilpi Goel, Marijn Heule, Matt Kaufmann, J Moore, Rob Sumners, Ivan Sutherland, Sol Swords, and my PhD students

October, 2019

Computer Sciences Department
University of Texas
2317 Speedway, M/S D9500
Austin, TX 78712-0233

hunt@cs.utexas.edu
TEL: +1 512 471 9748
FAX: +1 512 471 8885

Centaur Technology, Inc.
7600-C N. Capital of Texas Hwy
Suite 300
Austin, Texas 78731

hunt@centtech.com
TEL: +1 512 418 5797
FAX: +1 512 794 0717

Our Collaboration with Government and Industry

AMD ARM CenTaur Technology

GE intel IBM

ORACLE Rockwell Collins NATIONAL SECURITY AGENCY UNITED STATES OF AMERICA

Our own research includes:

- Development of **core technologies**
- **Application** of these technologies in different domains
- **Commercial Use:** Validation of processor designs

What is Formal Hardware Verification?

Demonstrating that design models have desired functional properties:

- Combinational circuit implements some function (e.g., addition)
- Sequential circuit satisfies some property (e.g., no deadlock)
- A design implements a specification (e.g., cooperating FMs implement an ISA)
- User-facing physical realization provides predictable and secure operation

To assure desired functionality, one must also account for:

- Circuit delays – depends on implementation technology
- Power consumption – concerns the size, speed, and work
- Transient errors – from energy events

We present a simple example, common contemporary features, validation procedures and mechanisms, and future needs.

The Verification Problem

- Different specification languages are used at each level.
  - Systems: Linux, Web, Internet, Storage Systems
  - Lanugages: C, C++, Java, Perl, Swift, models
  - ISA: x86, ARM, PowerPC, RISC-V, ...
  - Architecture: Drawings, Charts, Graphs, Natural Language
  - Microarchitectures: More diagrams, charts, etc.
  - Register-transfer: VHDL, Verilog
  - Netlist: VHDL, Verilog
  - Transistor Schematic: “Stick diagrams”
  - Layout: Colored Polygons
- The Size
  - C, C++ models: millions of pages
  - Binary models: billions of pages
  - ISA models: hundreds of pages
  - RTL models: tens of thousands of pages
  - Netlist models: millions of pages

When I Say Specification or Verification, I Mean

Specifying the operation of computing systems can be precise.

- Specifications are given as mathematical expressions.
- Implementation are also given as mathematical expressions.

By verification, I mean rigorous mathematical proof.

- Correctness proofs for computing systems are often very cost effective.

Don't get me wrong, I'm a realistic engineer.

- For instance, an architecture drawing conveys lots of information in a very compact form.
- Such documents are of great use for communicating a useful abstraction of system operation.
- But as a design specification such documents are insufficient.

A Typical Engineering Design Example

Consider the task of determining the maximum range of a rifle.

- Given that the end of the rifle muzzle is at ground level, what muzzle inclination angle (between 0 and $\pi/2$) gives the maximum range.
- Our analysis is unrealistically simple: no air resistance, flat plane, etc.

$$t = V_0 sin(\theta)/g \tag{1}$$

$$d = 2V_0 cos(\theta)t \tag{2}$$

$$d = \frac{2{V_0}^2 cos(\theta)sin(\theta)}{g} = \frac{{V_0}^2 sin(2\theta)}{g} \tag{3}$$

To discover the max distance, we take the derivative with respect to θ.

$$0 = cos(2\theta) \tag{4}$$

The solution of $\pi/4$ can be determined without any testing! In fact, the solution works for any initial velocity and any gravitational value.

A Simple Computer Design Problem Example

Consider a design problem.

- Specification: design a circuit that doubles a number.
- Solution: use an "available" adder and connect the number to be doubled into both of the adder inputs.
- Verification: hand-wave argument.

How can such a simple problem not have an obvious solution?

- What is wrong?
- Do we lack the mathematics?

We can "overload" algebra to model our intent and design.

- Specification: $2i$
- Implementation: $i + i$

We can prove the correctness of our design by mathematical proof.

$$\begin{aligned} i + i &= 2i \\ i + i &= i + 1i \\ i + i &= i + i + 0i \\ i + i &= i + i + 0 \\ i + i &= i + i \end{aligned}$$

This verifies our implementation for all i.

Super Simple Adder Verification

Consider a simple, 64-bit adder design – it should be trivial to verify. But, is it?

There are 128 Boolean inputs, so it requires 2^{128} tests.

- That's 340,282,366,920,938,463,463,374,607,431,768,211,456 tests!
- We don't perform this kind of comparisons; we compare the specification equations to design equations.
- We can perform such an adder comparison in milliseconds.

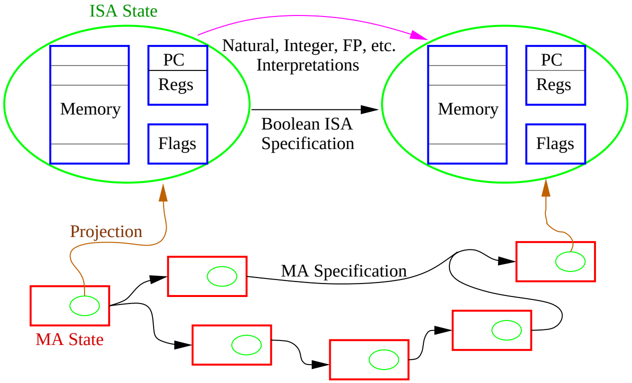
Our point? It's often much faster to do analysis using algebra than attempting exhaustive simulation.

We have scaled this approach to entire microprocessors.

And, we do the same thing with (compiled) binary programs.

Is Complete Netlist Verification Possible?

When a design is simplified to a netlist, it is possible to mechanically verify that a transistor- or gate-level netlist of a processor design meets its abstract functional specification.



The FM9001 Microprocessor

The FM9001 is a general-purpose, 32-bit microprocessor.

- The FM9001 ISA is formally specified as an ISA-level interpreter.
- The FM9001 design (including its test logic and I/O interface) was formally described in using the formally-defined `DE` HDL.
- The FM9001 was mechanically proven to meet its specification.
- The design was mechanically translated into LSI Logic's NDL.
- Test vectors were shown to detect all (but one) stuck-at faults.
- The FM9001 was manufactured by LSI Logic.
- The FM9001 was tested extensively without ever observing an error.

Developing a fully, formally-verified microprocessor design is possible.

So, what's the commercial story?

ARM, PowerPC, RISC-V, x86, and many others

There are many ISAs, and these variants target specific markets.

- Servers, workstations, and laptops use x86 (AMD, Intel, VIA)
- Mobile, tablets use ARM and RISC-V solutions, but many vendors
- No standards for special-purpose (graphics, network, auto) systems

Are commercial processor vendors using formal methods?

- Yes: ARM, Boeing, Centaur, General Electric, Intel, IBM, Oracle, Rockwell-Collins
- FM research results appear in CAV, FMCAD, CADE, ITP, DAC, ...

Commercial users primarily use BDD- and SAT-based equivalence checking, STE, model checking, and SMT.

In some cases, users steer theorem-proving systems to verify invariants, and to compose observations made with more automatic techniques.

Contemporary Example

Recently, Oracle developed floating-point divider and square-root function by using fast multiplier; guess made, answer improved, repeat 4 times.

- On a 1000+ machines, DIV and SQ test suite took a month+ to run
- As a result, design changes were carefully considered
- And, bugs still appeared...

Decided to use ACL2, hired a graduate from our group.

- Used our tools to read design into ACL2
- Developed proof (finding bugs), complete verification runs in hours
- Thus, design iteration cycle could be hours
- Shrunk *guess* ROM to 40% of original; made fixes so it worked
- Increased ROM size to 60% of original; reduced repeats to 3 times

Oracle's new FP RTL could be checked in hours – **no fear** design iterations occurred daily

This approach completely changed their floating-point design flow. At Centaur, these proofs only take minutes.

Post-Silicon Debug

Around 2000, IBM had designed and begun manufacturing of a multi-core, Power PC product. In systems test, with dozens of machines running, there were some machines that *locked up*.

- Couldn't reproduce problem reliably – took days to show itself
- Couldn't reproduce problem with simulation – far too slow
- \$100+ million of product *stuck* on factory floor

IBM contact formal methods group

- Working from the design (netlist), FV engineers were able to exhibit a trace of 120 steps that also caused a lockup
- This effort took weeks, while all other debugging efforts continued
- New silicon had to be manufactured
- New silicon didn't exhibit the problem
- Never determined if the bug discovered was the actual problem, but it changed something – and the problem disappeared!

These are powerful, post-silicon debug tools.

Contemporary Hardware Design

Companies specify, design, manufacture, and purvey, the largest and most complex computer-hardware artifacts, such as the x86.

Such commercial hardware offerings include:

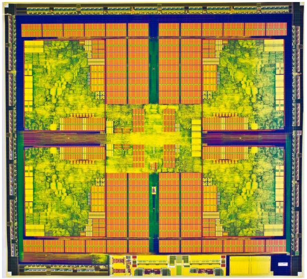
- Enormous ISAs with tremendous complexity
- Test logic; accounts for 5% – 10% of the final product
- Microcode programs (50K, 100+ bit) for initialization and exceptions
- Monitoring, CPU management, with an embedded processor
- Configuration mechanisms, such as fuses and MSRs
- Megabytes of internal memory
- Purpose-built, multi-channel memory interfaces (up to 8 channels)
- Timers and interrupt controllers
- I/O interfaces, such as Ethernet, USB, SATA, PCI-Express, ...
- Service interfaces, for in-field updates (e.g., recent Xeon bug)
- Special modes, registers, and hardware interfaces for debug

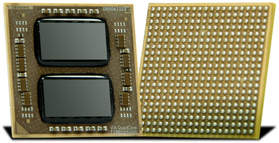
Contemporary processors *boot* themselves; involves *decompressing* microcode, clearing 1000s of registers, initializing memory system, etc.

VIA QuadCore Processor

Contemporary Example

- Full X86-64 compatible four-core design
- 14nm technology, 1+ billion transistors, core is only $2mm^2$
- AES, DES, SHA-1/256/384/512, and random-number generator
- Built-in security processor
- Runs 40 operating systems, four VMs





The Size of a Contemporary x86 Design

Industrial processor specifications provides more than the ISA.

- Designs are (usually) specified in Verilog.
- Designs are specified at the micro-architectural level.
- Design specifications also must include a simulation environment.

Without environment, Centaur's x86 specification is 1.4M lines of Verilog

Centaur's 4-core x86 design 1B+ transistors; single core is less than 2mm^2.

To read and process the specification:

- Before model build, run tests with four Verilog simulation systems.
- Takes ACL2 nine minutes to read Verilog and build its model: 2500 modules, 50 packages, multiple classes, 1700 parameter declarations
- Unparameterize; resolve ranges; make wire declarations explicit; eliminate some operators (++, –, ...), and many other simplifications
- Thousands of little proofs (checks) are done during the build.
- We find syntactic bugs – even after other simulators have run!
- We find functional bugs regularly.

What is the Specification?

Contemporary processor architectures (e.g., the x86) are specified with natural language, charts, graphs, tables, etc.

- AMD, Intel x86 customer-oriented documentation exceeds 3000 pages
- But, it's nowhere close to sufficient to build a working x86 processor
- There are 1000s of additional requirements held close by x86 vendors

Shilpi Goel (primary author), Warren Hunt, and Matt Kaufmann have built an ACL2-based x86 specification that includes:

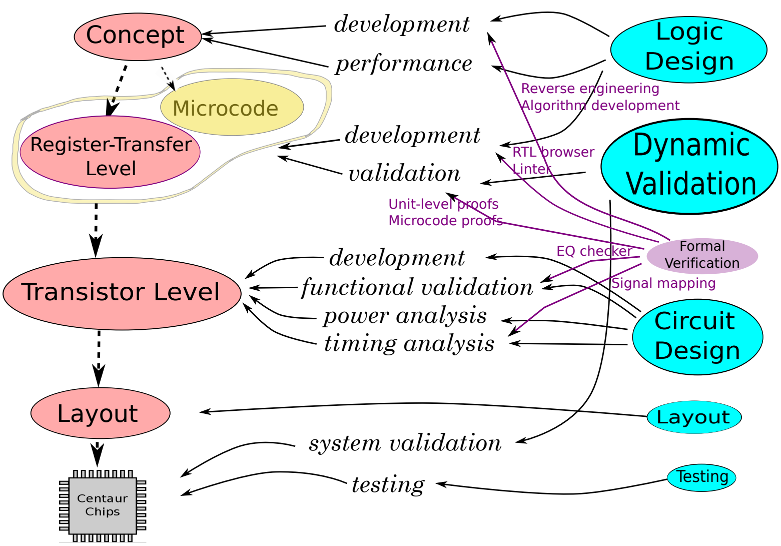
- Modeling focus: x86 64-bit (Intel's IA-32e) uniprocessor
- Opcodes: 413 (user and system mode instructions)
- Specification of paging and segmentation
- Specified system state, e.g., Local and Global Descriptor Tables
- User and system mode operation; system program verification possible
- Concrete execution: 300,000 (system) to 3.3 million (user-only) IpS
- Support for FV and debugging/dynamic instrumentation of x86 binary
- Automatically generated documentation for users and developers
- ACL2 x86 spec: 60K LoC (without macro expansion), about 240 files

Our X86 ISA Specification

Our x86 specification:

- Is a compile-to specification
- Is a build-to specification
- Is a formal simulator of the x86 ISA
- Provides the semantics for verifying x86 machine code
- Provides a model that supports symbolic execution of x86 programs
- Has been used to verify a zero-copy program; this involves paging and reasoning about tens of memory accesses per instruction
- Is being used by one x86 vendor

Design Flow, Augmented with Formal Verification



Example Properties Mechanically Verified

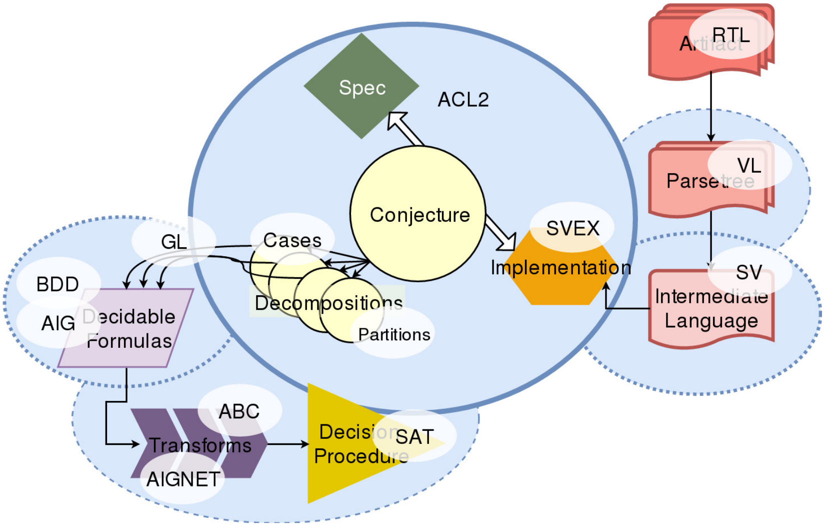
Centaur has specified 1000s of properties for their x86 implementation

- Upon design update, properties (proofs) are (re-)checked using 100+ machines
- Failures automatically reported to the FV group. Why the failure?
  - Spec error, Tool failure (capacity, orchestration, ...), new *feature*
  - Actual error in design
- Determine root cause of error; if actual bug, designer alerted

What kind of properties are checked?

- Data functional units - 1000+ instructions
  - Integer, media, floating-point, graphics
  - On a clock-by-clock basis, pipelined multipliers are reconfigured to perform different sizes and types of multiplications – quite complex
  - string decompression (800 microinstructions), several nested loops
  - 128-by-64-bit division (50 microinstructions), and so on...
- Memory system properties
  - Memory system is the most complex part of a modern design
  - Verify invariant properties; e.g., atomic read-modify-write memory
- Analyze clock trees & synthesis results; reverse-engineer timing paths

Verification Flow



So What?

These techniques save time, improve communications, reduce costs...

- Question: can our approach be valuable to your organization?
- Answer: **Yes!** especially when integrated into the design flow so that verification results arrive in minutes or, at most, hours.
- The cost savings are obvious. Our tools have found thousands of bugs — many of which might have otherwise shipped.
- Improves accuracy of specifications, reduces errors, provides specifications that can be analyzed and simulated.
- Design iteration becomes much faster. And, as a design grows, our techniques scale – proofs can processed in parallel.
- Also used for post-silicon debug, fast simulation, and reverse-engineering.
- Result: accurate specs, proof-based validation, faster design cycles,

Saves financial and human capital, reduces re-work, yields better products.

The Size of The Future

New x86 contains I/O and memory interface – size of everything is much bigger!
Remember: we are dealing with very large models; this is different that checking 1000s (or billions) of simple (SAT-checked) properties.

- Multiple name spaces, e.g., Verilog Classes (ACL2 packages)
- Spec: now 1.5M LoC; future; 20M+ LoC with I/O, network, etc.
- Proofs: now some proofs 40+ GB RAM, require 150+ CPU hours; future: 100s GBs RAM, 1000s of CPU hours
- Memory compression is *critical* for spec and proofs (see next slide)!
- Automation++: purpose-built proof procedures (clause processors)
- Hundreds of changes/improvements to improve ACL2's capacity!

Tool capacity can't be overstated; once x86 design translated into ACL2 representation and stored, it can be read (by many machines) in seconds.

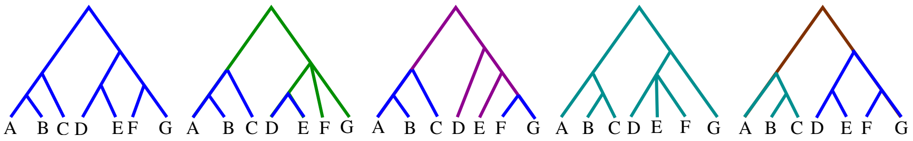
Model Representation

It is critical to keep the model compact and up-down accessible.

- The VIA x86 model is 1.5M LoC; $\sim$60 MBytes. Processing requires 9 minutes.
- Once processed, model is expanded (elaborated) and annotated.
- Every ACL2 expression is unique – no duplicates anywhere!
- With compression, entire model can be read off disk in seconds.
- Upward map created (more later); this allows netlist to up/down navigated.
- Unique representation of all netlist data speeds comparisons greatly.
- Model can be regurgitated in a *simplified* form; used for IP study.

The compressed model is a starting point for interfacing to many other tools.

Model Is Like a Up-Down DataBase



Netlist-Level Representation

```
(((A B) C) ((D E) (F G)))
(((A B) C) ((D E) F G))
(((A B) C) (D (E (F G))))
((A (B C)) ((D E F) G))
((A (B C)) ((D E) (F G)))
```

Internal size ∼3/5; much smaller (<10%) for a big model.

```
((\#1=((A B) C) \#3=(\#2=(D E) \#4=(F G)))
(\#1\#(\#2\# F G))
(\#1\#(D (E \#4\#)))
(\#5=(A (B C)) ((D E F) G))
(\#5\#\#3\#))
```

Can navigate model in up-down manner. Association list is the *up* map!

(**(A B)** ((A B) C))
(**(((A B) C) ((D E) (F G)))** . 1)
(**(D E)** ((D E) F G)
((D E) (F G)))
(**((D E) F G)** ((A B) C) ((D E) F G))
(**(((A B) C) ((D E) F G))** . 1)
(**((A B) C)** (((A B) C) (D (E (F G))))
(((A B) C) ((D E) F G))
(((A B) C) ((D E) (F G))))
(**(F G)** (E (F G))
((D E) (F G)))
(**(E (F G))** (D (E (F G))))

(**(D (E (F G)))** (((A B) C) (D (E (F G)))))
(**(((A B) C) (D (E (F G))))** . 1)
(**(B C)** (A (B C)))
(**(D E F)** ((D E F) G))
(**((D E F) G)** ((A (B C)) ((D E F) G)))
(**((A (B C)) ((D E F) G))** . 1)
(**(A (B C))** ((A (B C)) ((D E) (F G)))
((A (B C)) ((D E F) G)))
(**((D E) (F G))** ((A (B C)) ((D E) (F G)))
(((A B) C) ((D E) (F G))))
(**((A (B C)) ((D E) (F G)))** . 1)

The Future

We believe these techniques are the engineering tools of the future.

- Our approach is more much more thorough than simulation and requires far fewer computational resources.
- Ability to combine HW and SW models, including environments
- Tools inspired by our approach are now being used to help assure the validity of financial trading
- Key issue: tool flexibility. CAD Vendors have greatly slowed progress. No one owns their software – it's nearly all leased!
- Formal specification and proof of IP is possible; need a way monetize and share, including the formal specs, designs, and the proofs.
- Simply said, mathematical approach is more precise, faster, assures better products, and is less costly than provided by the existing practice.

Mechanized mathematics is the only approach that can scale to future system specification and verification requirements.