Industrial Use (of ACL2) for Hardware Verification

Warren A. Hunt, Jr.

Joint work with: Bob Boyer, Marijn Heule, Matt Kaufmann, J Moore, and many others (identified later in talk)

June, 2017

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

Ecosystem

We collaborate with government and industry.

Our own research includes:

- Development of **core technologies**
- **Application** of these technologies in different domains
- **Commercial Use:** Validation of processor design (∼10 FV personnel, 20-30 additional users) at CenTaur echnology and ORACLE

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

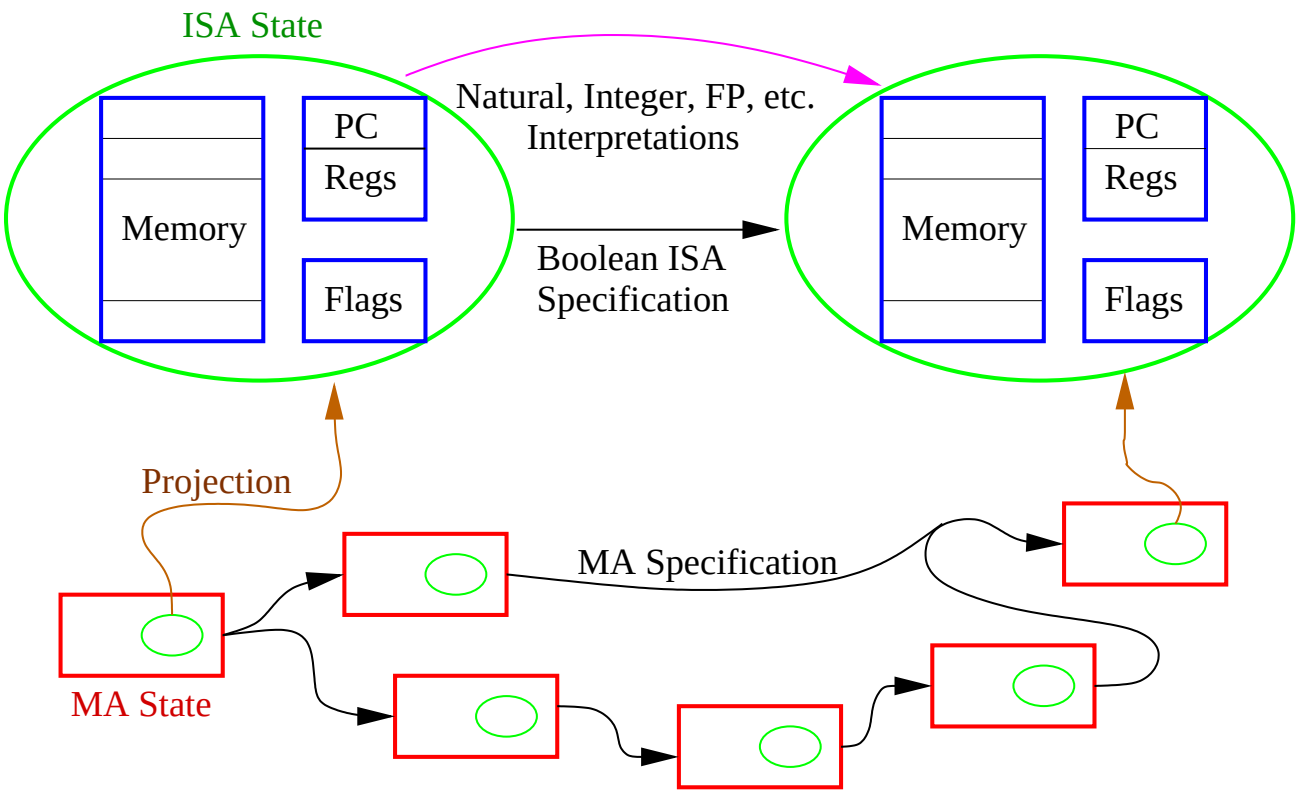
To assure desired functionality, one must also account for:

- 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.

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 `DUAL-EVAL` HDL.
- The FM9001 was mechanically proven to meet its specification.
- The design was mechanically translated into LSI Logic's NDL.
- Test vectors were created and proven 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, x86, and many others

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

- Servers, workstations, and laptops use x86 (AMD, Intel, VIA)
- In mobile devices and tablets, the ARM ISA (but not ARM Ltd) dominates
- There is a wide variety of embedded (networking, appliance, automotive, medical) systems.

Are commercial processor vendors using formal methods?

- Yes: ARM, Centaur, General Electric, Intel, IBM, Oracle, Rockwell-Collins
- FM research results appear in CAV, FMCAD, and other conferences

All commercial users primarily use BDD- and SAT-based equivalence checking, STE, and model checking.

Users steer the ACL2 theorem prover to also verify invariants and to compose observations made with more automatic techniques.

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:

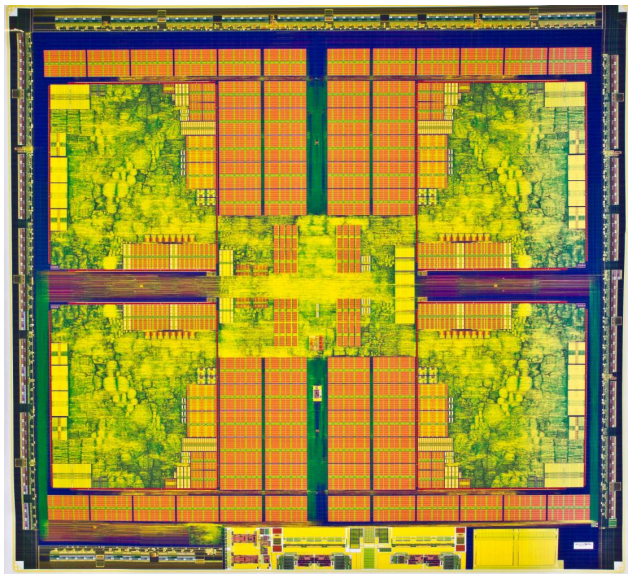
- Enormous ISAs with tremendous complexity
- Test logic; accounts for 5% – 10% of the final product
- Microcode programs (50K, 100+ bit) for initialization and exceptions
- Monitoring, CPU management, with an embedded processor
- Configuration mechanisms, such as fuses and MSRs
- Megabytes of internal 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

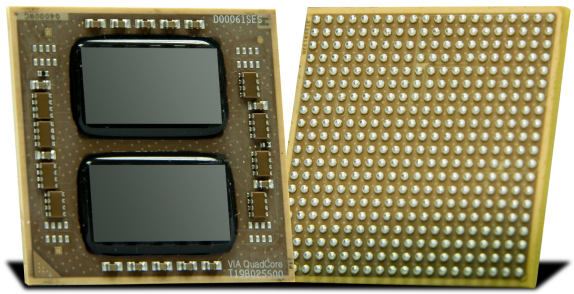
Contemporary processors *boot* themselves; involves *decompressing* microcode, clearing 1000s of registers, initializing memory system, etc.

VIA QuadCore Processor

Contemporary Example

- Full X86-64 compatible four-core design
- 28nm technology, 227.6 million transistors
- 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

In industry, processor specifications are much lower level than our specification.

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

Centaur's current x86 design is 1.4M lines of Verilog, exclusive of its environment.

Centaur's current 4-core x86 design exceeds 1B transistors

To read and process the specification

- Before model build, run tests with four Verilog systems.
- Takes ACL2 about five minutes to read Verilog and build its model.
- Thousands of little proofs (checks) are done during the build.
- We often find syntactic bugs – even after simulations are run!
- We regularly find functional bugs.

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 has built an ACL2-based x86 specification that includes:

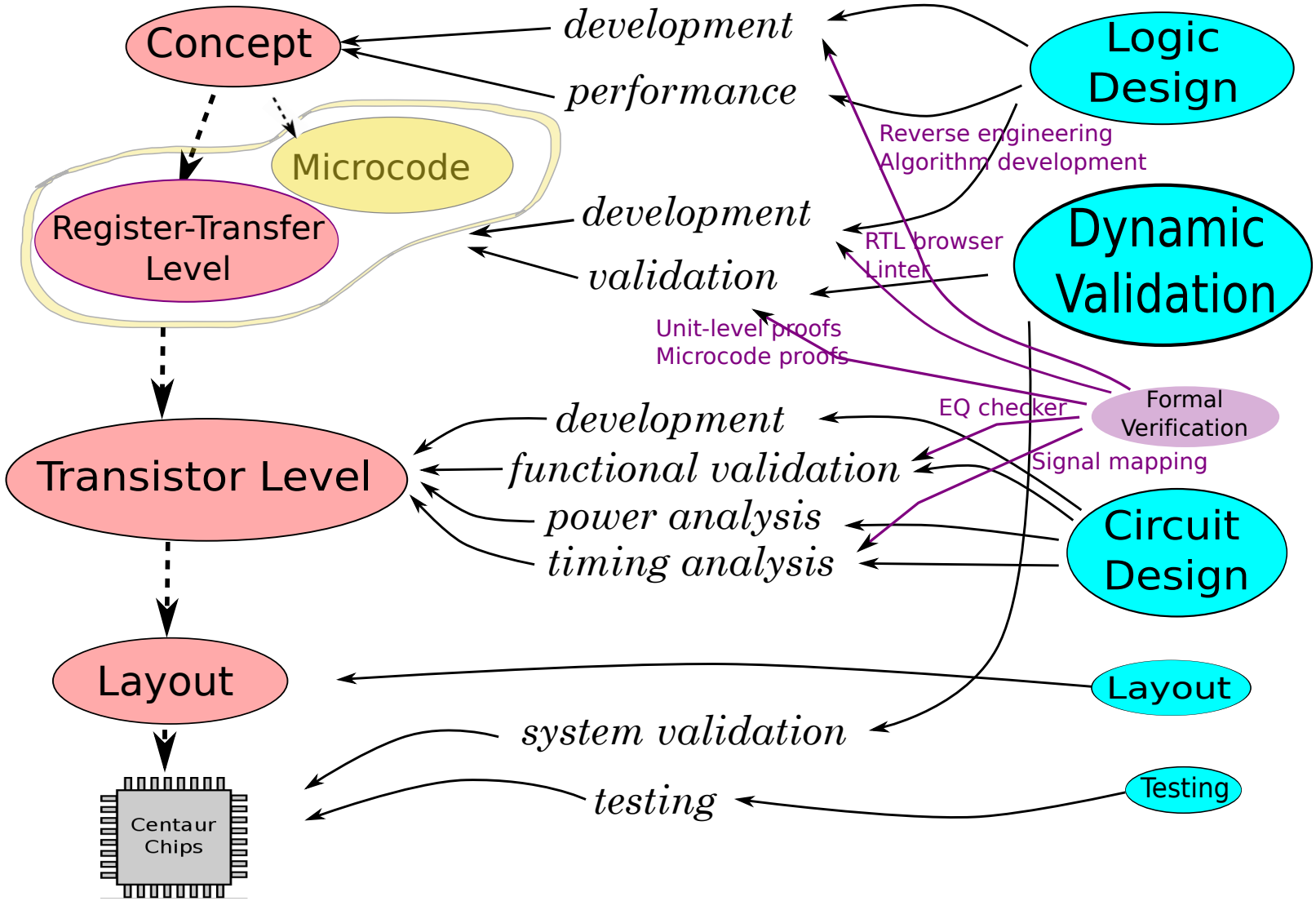
- Modeling focus: x86 64-bit (Intel's IA-32e) uniprocessor
- Opcodes: 413 (user and system mode instructions)
- Specification of paging and segmentation
- Specified system state, e.g., Local and Global Descriptor Tables
- User and system mode operation; system program verification possible
- Concrete 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: 60,000 LoC (without macro expansion), 240 files

Our X86 ISA Specification

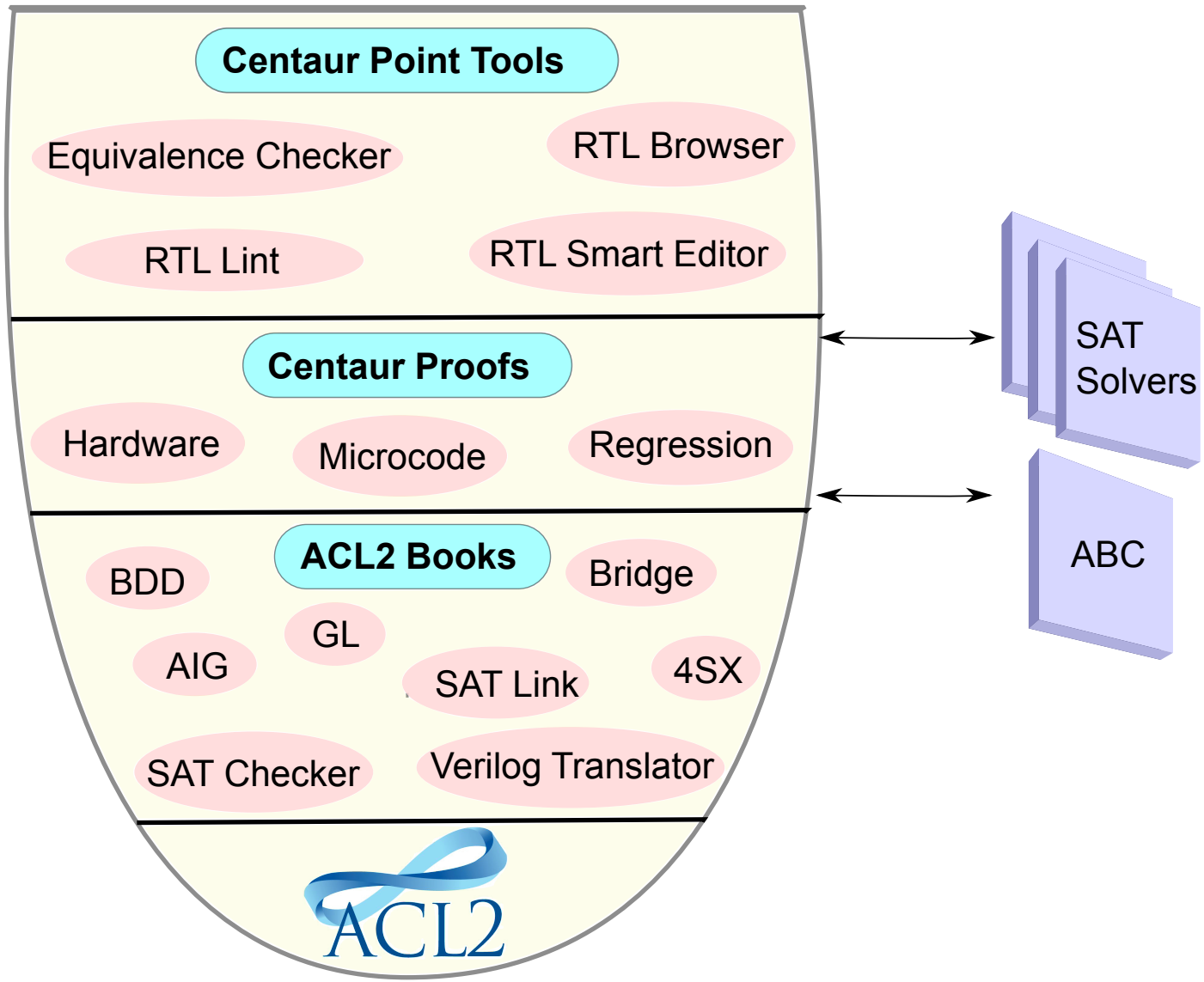
Our x86 specification:

- 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



Verification Framework



Example Properties Mechanically Verified

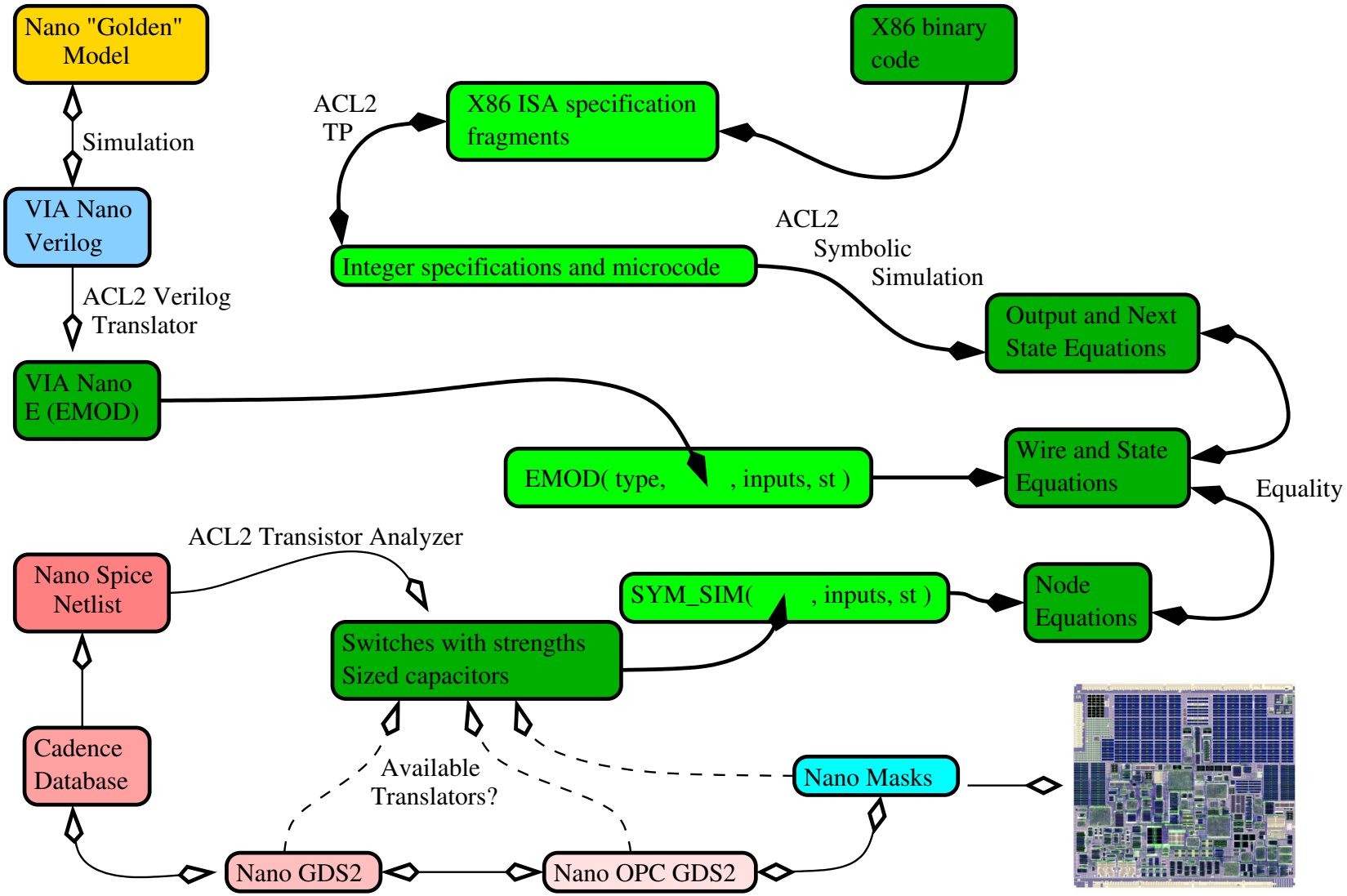
Centaur has specified 1000s of properties for their x86 implementation

- Every night, all properties (proofs) are checked using 100+ machines
- Failures automatically reported to the FV group. Why the failure?
  - Specification error
  - Tool failure (capacity, orchestration, ...)
  - 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
  - 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

The Centaur Verification Tool Relationships



Groups that sometimes use ACL2 with Whom We Interact

ARM: *David Russinoff*

Apple: *Jared Davis*

Centaur: *Shilpi Goel*, Anna Slobodova, *Rob Sumners*, *Sol Swords*

Galois: *Ben Selfridge*

General Electric: Harsh Chamarthi, *Pete Manolios* (also of NEU)

Intel: *Nathan Wetzler*

IBM: *Bishop Brock*, *Jun Sawada*

Kestrel: Alessandro Coglio, Eric McCarthy, *Eric Smith*, Stephen Westfold

Oracle: Andrew Brock, Jo Ebergen, *Keshav Kini*, *David Rager*

PSU: Chris Chen, Chris Cowen, Marly Roncken, Ivan Sutherland

Rockwell-Collins: Dave Greve, David Hardin

UBC: Mark Greenstreet, Yan Peng

UT (our group): *Cuong Chau, Mihir Mehta, Mertcan Temel, Bill Young*

Charge by the JASON Group – with Some Responses

1. Which national security challenges could benefit most from accelerating development of industrial-scale formal methods? For which challenges might such methods provide new strategic capabilities?

Actually procuring and fielding what is mathematically specified. Exact exchange of specifications between multiple vendors and the Government. Security – anywhere there is a flaw, there is an exploit *opportunity*.

Enormous cost reductions.

2. How might we quantify the impact of these approaches to national security challenges? What impact can we expect to achieve?

As compared to existing methods, cost reduction for product/capability specification, design, validation, verification, deployment, operation, and maintenance.

Charge by the JASON Group – with Some Responses

3. What are the impediments to the broad and successful use of industrial-scale formal methods? Are there strategies to mitigate this impediments?

Trained people, lack of mathematically-based computing processes, general ignorance of what is now possible, lack of commercial FV tool providers supporting publicly-available FV languages.

Industrial-strength FV specification and analysis systems.

Hardware companies have been driven to use FV for cost reasons; they can't test in quality. The same argument can be made for software, but the cost/benefit and engineer mentality isn't as obvious.

Charge by the JASON Group – with Some Responses

4. What are new/promising approaches to consider in formal methods for addressing the national security challenges?

I am a practical engineer. I know that many, many things will neither be specified nor analyzed with FV techniques. But, for those areas where we have already demonstrated reduced cost, fewer errors, and shorter time-to-completion, we should accelerate the use of FV.

Note: FV isn't a *hot* topic. Machine learning, data mining, and AI are (currently) hot topics, but they need FV too.

Unless we formalize these topics and verify that the inferences derived (as determined by implementation software) are correct, we will just have increasingly *noisy and inaccurate* information systems. For example, we know of no formal specification for neural nets.

FV offers a means to build more reliable systems; we should embrace it.

Charge by the JASON Group – with Some Responses

5. What is the technology base (e.g. workforce, education, software, hardware, tools, etc.) needed to support industrial scale formal methods? Is it sufficient? Are new/special capabilities needed?

The USA has a few hundred trained FV people. Europe probably has a thousand or more. Other countries don't have many trained people.

Projects are not started because one can use FV. But, FV can help provide a rigorous engineering process.

When the USA built the *bomb* and went to the moon, we had to train a lot of people, develop processes, and harden such processes so they could be used at scale.

If the USA wants reliable computing and communications for military, medical, transportation, commerce, and financial needs, we'd better learn how to specify what we want and have tools available that are capable of assuring that proposed solutions do indeed provide the desired properties.

Charge by the JASON Group – with Some Responses

6. What are the national security consequences for not accelerating the development of our capabilities in industrial scale formal methods?

Losing the ability to accurately manage computer-based information systems will cause us to lose the information battle, which, in turn, threatens our very way of life.

Our ability to project force, both kinetic and financial, depends critically on reliable, precise computing capabilities.

Our ability to protect our C3 and C4 infrastructure depends on our ability to assure (and even prove) that our systems are safe from attack.

Charge by the JASON Group – with Some Responses

The questions above have been posed to JASON by the study sponsor and JASON's response to these questions will be a key part of the study report.

In addition to the questions posed above, there are broad issues that pertain to the potential enhanced use of formal methods:

1. Is now a good time to consider enhanced use of formal methods? If not, what are the issues preventing expanded application of these methods?

It's past time. Each day we delay is another day of exposure to some unnecessary risks.

Charge by the JASON Group – with Some Responses

2. Are we asking the right questions regarding securing software and hardware? Are there approaches that should be considered apart from or in addition to formal methods?

In the 20th century, continuous mathematics caused science to explode.

In the 21th century, computer-based systems will be used everywhere. We need suitable mathematics to specify and verify computer-based artifacts.

We need mechanized mathematics to analyze the already extremely-large computer systems that have become the foundation of our information-based society.

Charge by the JASON Group – with Some Responses

3. Can one assess the level of security of hardware or software that can be achieved using such methods? We welcome the speakers' perspectives on any of the issues raised above.

FV can help assess hardware and software will behave as specified. If such systems have been mechanically checked to work as specified, then we can trust the top-level specifications to accurately reflect system implementation properties.

We can independently analyze top-level specifications.

FV provides a means to analyze models and to assure that implementations of models are sound (within the limitations of the models themselves).

Two Observations by Bob Boyer

I have heard that civil engineering was not licensed in California until a dam broke and killed 900 people. Today, anyone can program anything they want. Do we need engineering schools to teach what is needed so that software people can pass licensing exams, preferably exams which deprecate the 'let the customers find the bugs' attitude, and which emphasize taking mathematics everywhere in the software process?

The product liability laws of our nation, in the case of computer systems, seem based upon 'let the customers find the bugs, and then we will try to fix them when we can'. The computer revolution is over 50 years old. Is it time to look at product liability legislation? Maybe that would help corporate officers take a harder look at formal methods.