Formal Assurance for RISC-V Implementations

Daniel M. Zimmerman and Joseph R. Kiniry

8th RISC-V Workshop, Barcelona, Spain – 9 May 2018

This material is based upon work supported by the Defense Advanced Research Projects Agency (DARPA) under Contract No. HR0011-18-C-0013. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the Defense Advanced Research Projects Agency (DARPA).

Approved for Public Release, Distribution Unlimited

About Galois

- established in 1999 to apply functional programming and formal methods to the problem of information assurance

© 2018 Galois, Inc.

About Galois

- established in 1999 to apply functional programming and formal methods to the problem of information assurance
- over 40 active projects for numerous clients, both U.S. government (NSA, DARPA, IARPA, Homeland Security, Air Force Research Lab) and commercial (Amazon, others)

© 2018 Galois, Inc.

About Galois

- established in 1999 to apply functional programming and formal methods to the problem of information assurance
- over 40 active projects for numerous clients, both U.S. government (NSA, DARPA, IARPA, Homeland Security, Air Force Research Lab) and commercial (Amazon, others)
- over the years, we have substantially broadened our scope to *high assurance everything*

© 2018 Galois, Inc.

The Many Definitions of RISC-V

- RISC-V Instruction Set Manual defines the various flavors of the RISC-V ISA in **human-readable** fashion

 © 2018 Galois, Inc.

The Many Definitions of RISC-V

- RISC-V Instruction Set Manual defines the various flavors of the RISC-V ISA in **human-readable** fashion
- RISC-V has been **mechanized** in several traditional ways:
    - software simulation at various abstraction levels (e.g., gem5, Spike, RV8, Renode, QEMU, Imperas, Esperanto)
    - hand-written RTL in Verilog and SystemVerilog
    - hand-written HDL in Chisel and Bluespec SystemVerilog (BSV)

© 2018 Galois, Inc.

The Many Definitions of RISC-V

- RISC-V Instruction Set Manual defines the various flavors of the RISC-V ISA in **human-readable** fashion
- RISC-V has been **mechanized** in several traditional ways:
  - software simulation at various abstraction levels (e.g., gem5, Spike, RV8, Renode, QEMU, Imperas, Esperanto)
  - hand-written RTL in Verilog and SystemVerilog
  - hand-written HDL in Chisel and Bluespec SystemVerilog (BSV)
- ISA semantics working group has been mechanizing RISC-V **semantics** in both traditional and unusual ways, including:
  - multiple Haskell implementations (Bluespec, MIT, Galois)
  - mechanization in logical frameworks (MIT, with Coq)

© 2018 Galois, Inc.

Assurance

- all these different definitions/mechanizations raise an important question:

  how can we provide assurance about them?

© 2018 Galois, Inc.

Assurance

- all these different definitions/mechanizations raise an important question:

  how can we provide assurance about them?

- providing assurance about a computing system basically means proving two things:

© 2018 Galois, Inc.

Assurance

- all these different definitions/mechanizations raise an important question:

  how can we provide assurance about them?

- providing assurance about a computing system basically means proving two things:

  1. the system does what it is supposed to do (designed to do, specified to do)

© 2018 Galois, Inc.

Assurance

- all these different definitions/mechanizations raise an important question:

  how can we provide assurance about them?

- providing assurance about a computing system basically means proving two things:

  1. the system does what it is supposed to do (designed to do, specified to do)
  2. the system does not do anything else

© 2018 Galois, Inc.

Assurance

- all these different definitions/mechanizations raise an important question:

  how can we provide assurance about them?

- providing assurance about a computing system basically means proving two things:
  1. the system does what it is supposed to do (designed to do, specified to do)
  2. the system does not do anything else
- #2 is often overlooked!

© 2018 Galois, Inc.

RISC-V Assurance

- so, given an implementation that claims to be RV32I, how do we know that it *really is* RV32I?

© 2018 Galois, Inc.

RISC-V Assurance

- so, given an implementation that claims to be RV32I, how do we know that it *really is* RV32I?
- a typical answer: **run lots of conformance tests!**

© 2018 Galois, Inc.

RISC-V Assurance

- so, given an implementation that claims to be RV32I, how do we know that it *really is* RV32I?
- a typical answer: **run lots of conformance tests!**
    - passes the buck to test writers, who now need to prove that conformance tests are 100% complete and are really testing for RV32I-ness

© 2018 Galois, Inc.

RISC-V Assurance

- so, given an implementation that claims to be RV32I, how do we know that it *really is* RV32I?
- a typical answer: **run lots of conformance tests!**
  - passes the buck to test writers, who now need to prove that conformance tests are 100% complete and are really testing for RV32I-ness
  - doesn't take into account that an implementation might have undesirable behaviors (i.e., overlooks #2)

© 2018 Galois, Inc.

RISC-V Assurance

- a better answer: **formally verify it!**

© 2018 Galois, Inc.

RISC-V Assurance

- a better answer: **formally verify it!**
- for example:
  - compile a Haskell RV32I to RTL via Clash
  - use equivalence checking tools to relate that compiled RTL, piece by piece, to the RTL of the RV32I implementation you want assurance for

 © 2018 Galois, Inc.

RISC-V Assurance

- a better answer: **formally verify it!**
- for example:
  - compile a Haskell RV32I to RTL via Clash
  - use equivalence checking tools to relate that compiled RTL, piece by piece, to the RTL of the RV32I implementation you want assurance for
- both commercial and open source equivalence checking tools are available for Verilog/SystemVerilog
  - Clifford Wolf is doing this with Yosys
  - at Galois, we're experimenting with JasperGold and other tools

© 2018 Galois, Inc.

Unfortunately…

- equivalence checking for a shallow-pipeline in-order core RISC-V is one thing…

© 2018 Galois, Inc.

Unfortunately…

- equivalence checking for a shallow-pipeline in-order core RISC-V is one thing…
- deep-pipeline, multi-threaded, out-of-order implementations that may have several extensions and custom designs are an **entirely different world!**

© 2018 Galois, Inc.

Unfortunately…

- equivalence checking for a shallow-pipeline in-order core RISC-V is one thing…
- deep-pipeline, multi-threaded, out-of-order implementations that may have several extensions and custom designs are an **entirely different world!**
- equivalence checking tools currently only exist for Verilog/SystemVerilog…

 © 2018 Galois, Inc.

Unfortunately…

- equivalence checking for a shallow-pipeline in-order core RISC-V is one thing…
- deep-pipeline, multi-threaded, out-of-order implementations that may have several extensions and custom designs are an **entirely different world!**
- equivalence checking tools currently only exist for Verilog/SystemVerilog…
  - bye-bye higher-level HDLs (BSV, Chisel, even SystemC) — or at least all of their advantages, since you have to work at the level of the resulting Verilog to get assurance

© 2018 Galois, Inc.

A Way Forward

- our ongoing R&D applies our experience in applied formal methods for software to this problem

© 2018 Galois, Inc.

A Way Forward

- our ongoing R&D applies our experience in applied formal methods for software to this problem

- in order to reason about a RISC-V implementation we need:

© 2018 Galois, Inc.

A Way Forward

- our ongoing R&D applies our experience in applied formal methods for software to this problem
- in order to reason about a RISC-V implementation we need:
  - machine-readable specifications of the **correctness** and **security** of an implementation

© 2018 Galois, Inc.

A Way Forward

- our ongoing R&D applies our experience in applied formal methods for software to this problem

- in order to reason about a RISC-V implementation we need:
  - machine-readable specifications of the **correctness** and **security** of an implementation
  - a way to **measure** the conformance of an implementation to these specifications

© 2018 Galois, Inc.

A Way Forward

- our ongoing R&D applies our experience in applied formal methods for software to this problem

- in order to reason about a RISC-V implementation we need:
  - machine-readable specifications of the **correctness** and **security** of an implementation
  - a way to **measure** the conformance of an implementation to these specifications
  - ways to work with (**summarize**, **understand**, and **explore**) such measurements

© 2018 Galois, Inc.

Correctness Specifications

- many different specs of various kinds and levels of rigor/completeness:
  - the Foundation's RISC-V executable tests: https://github.com/riscv/riscv-tests
  - Clifford's reference Verilog implementation: https://github.com/cliffordwolf/riscv-formal
  - Thomas's RISC-V semantics in Haskell: https://github.com/mit-plv/riscv-semantics
  - Nikhil's RISC-V semantics in Haskell and BSV: https://github.com/rsnikhil
  - SRI-CSL's semantics in L3: https://github.com/SRI-CSL/l3riscv
  - Galois's RISC-V semantics in Haskell

© 2018 Galois, Inc.

Correctness Specifications

- many different specs of various kinds and levels of rigor/completeness:
  - the Foundation's RISC-V executable tests: https://github.com/riscv/riscv-tests
  - Clifford's reference Verilog implementation: https://github.com/cliffordwolf/riscv-formal
  - Thomas's RISC-V semantics in Haskell: https://github.com/mit-plv/riscv-semantics
  - Nikhil's RISC-V semantics in Haskell and BSV: https://github.com/rsnikhil
  - SRI-CSL's semantics in L3: https://github.com/SRI-CSL/l3riscv
  - Galois's RISC-V semantics in Haskell
- how can these specifications be used effectively?

© 2018 Galois, Inc.

Specification <u>Validation</u>

- *is this mechanized ISA specification a correct interpretation of a RISC-V Instruction Set Manual?*
- *rigorous validation* through execution

© 2018 Galois, Inc.

Specification Validation

- *is this mechanized ISA specification a correct interpretation of a RISC-V Instruction Set Manual?*

- *rigorous validation* through execution
  - *ad hoc testing*: execute programs to see if a mechanized ISA does what we expect (when viewed as an I/O relation)

© 2018 Galois, Inc.

Specification Validation

- *is this mechanized ISA specification a correct interpretation of a RISC-V Instruction Set Manual?*

- *rigorous validation* through execution
  - *ad hoc testing*: execute programs to see if a mechanized ISA does what we expect (when viewed as an I/O relation)
  - *simulation coverage analysis*: translate axiomatic properties of a mechanized ISA specification to a test bench, then measure coverage of those properties across validation runs

© 2018 Galois, Inc.

Specification Validation

- *is this mechanized ISA specification a correct interpretation of a RISC-V Instruction Set Manual?*

- *rigorous validation* through execution
  - *ad hoc testing*: execute programs to see if a mechanized ISA does what we expect (when viewed as an I/O relation)
  - *simulation coverage analysis*: translate axiomatic properties of a mechanized ISA specification to a test bench, then measure coverage of those properties across validation runs
  - *bisimulation*: execute at least two mechanized ISA specifications or implementations and/or two semantically equivalent programs to pointwise compare their behavior (cf. Bluespec RISC-V Verification Factory)

© 2018 Galois, Inc.

Specification Verification

- *is this mechanized ISA specification a correct interpretation of a RISC-V Instruction Set Manual?*

- *rigorous verification* through formal reasoning

© 2018 Galois, Inc.

Specification Verification

- *is this mechanized ISA specification a correct interpretation of a RISC-V Instruction Set Manual?*

- *rigorous verification* through formal reasoning
  - *test benches as verification artifacts*: formally reason about test benches — prove that a mechanized ISA does what we expect, by proving that each test always passes

 © 2018 Galois, Inc.

Specification Verification

- *is this mechanized ISA specification a correct interpretation of a RISC-V Instruction Set Manual?*

- *rigorous verification* through formal reasoning
    - *test benches as verification artifacts*: formally reason about test benches — prove that a mechanized ISA does what we expect, by proving that each test always passes
    - *verification coverage analysis*: formally reason about properties specified in a test bench, measure coverage of those properties across verification runs

© 2018 Galois, Inc.

Specification Verification

- *is this mechanized ISA specification a correct interpretation of a RISC-V Instruction Set Manual?*

- *rigorous verification* through formal reasoning
  - *test benches as verification artifacts*: formally reason about test benches — prove that a mechanized ISA does what we expect, by proving that each test always passes
  - *verification coverage analysis*: formally reason about properties specified in a test bench, measure coverage of those properties across verification runs
  - *bisimulation*: prove that at least two mechanized ISA specifications or implementations are equivalent when viewed as an equivalence relation over traces

© 2018 Galois, Inc.

Security Specifications

- security properties have a different “flavor” than correctness properties — they often take the form “the following property must never hold”

© 2018 Galois, Inc.

Security Specifications

- security properties have a different “flavor” than correctness properties — they often take the form “the following property must never hold”
- Galois is developing…

© 2018 Galois, Inc.

Security Specifications

- security properties have a different "flavor" than correctness properties — they often take the form "the following property must never hold"
- Galois is developing…
  - a domain specific language (DSL) called LANDO that lets you specify…

© 2018 Galois, Inc.

Security Specifications

- security properties have a different “flavor” than correctness properties — they often take the form “the following property must never hold”
- Galois is developing…
  - a domain specific language (DSL) called LANDO that lets you specify…
    - the architecture,

© 2018 Galois, Inc.

Security Specifications

- security properties have a different “flavor” than correctness properties — they often take the form “the following property must never hold”
- Galois is developing…
  - a domain specific language (DSL) called LANDO that lets you specify…
    - the architecture,
    - correctness properties, and

© 2018 Galois, Inc.

Security Specifications

- security properties have a different “flavor” than correctness properties — they often take the form “the following property must never hold”
- Galois is developing…
  - a domain specific language (DSL) called LANDO that lets you specify…
    - the architecture,
    - correctness properties, and
    - security properties of a hardware design

© 2018 Galois, Inc.

Security Specifications

- security properties have a different "flavor" than correctness properties — they often take the form "the following property must never hold"
- Galois is developing…
    - a domain specific language (DSL) called LANDO that lets you specify…
        - the architecture,
        - correctness properties, and
        - security properties of a hardware design
    - a RISC-V security test suite that lets you execute a set of tests to roughly measure security and evaluate coverage

© 2018 Galois, Inc.

Measurement and Metrics

- in order to measure and judge the qualities of a design or implementation, we need metrics!
  - *power*: energy use while running a test bench
  - *performance*: execution speed of a test bench
  - *area*: design complexity measurements and layout size estimates based upon area use of comparable circuits
  - *security*: execution or reasoning about a security test bench to roughly or precisely measure what kind of vulnerabilities are mitigated

© 2018 Galois, Inc.

Working with Measurements

- we must help systems engineers **understand** and **explore** the effects of design and implementation decisions on power, performance, area, and security

© 2018 Galois, Inc.

Working with Measurements

- we must help systems engineers **understand** and **explore** the effects of design and implementation decisions on power, performance, area, and security
- *dashboard*: a Tufte-inspired **visualization** of a design, its implementations, and their critical characteristics

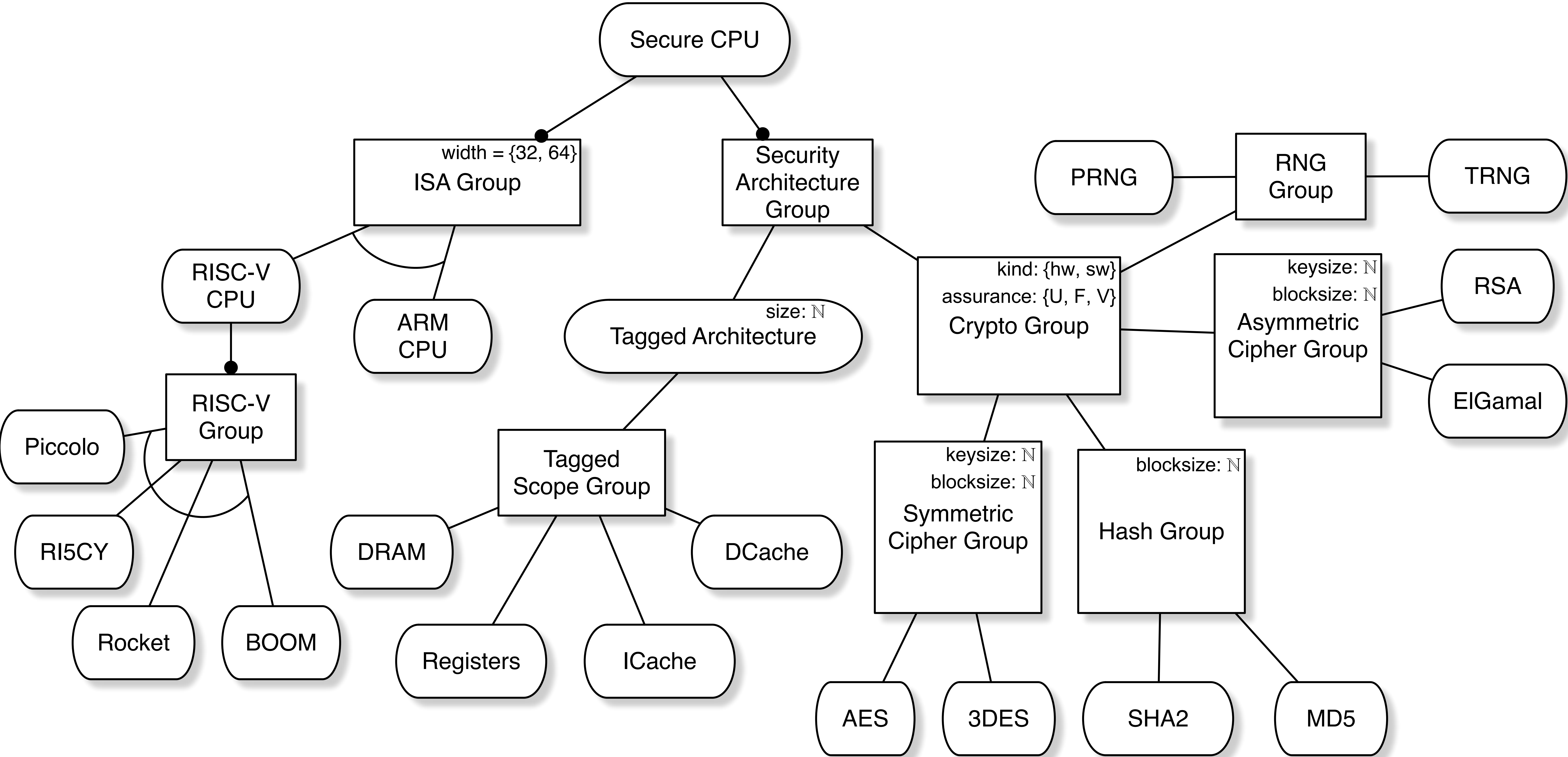
© 2018 Galois, Inc.

Working with Measurements

- we must help systems engineers **understand** and **explore** the effects of design and implementation decisions on power, performance, area, and security
- *dashboard*: a Tufte-inspired **visualization** of a design, its implementations, and their critical characteristics
- *product line engineering*:
  - view a hardware design as a **product line**
  - enable exploration of/reasoning about products derived from a formal model and its design via an **automatically generated feature model**

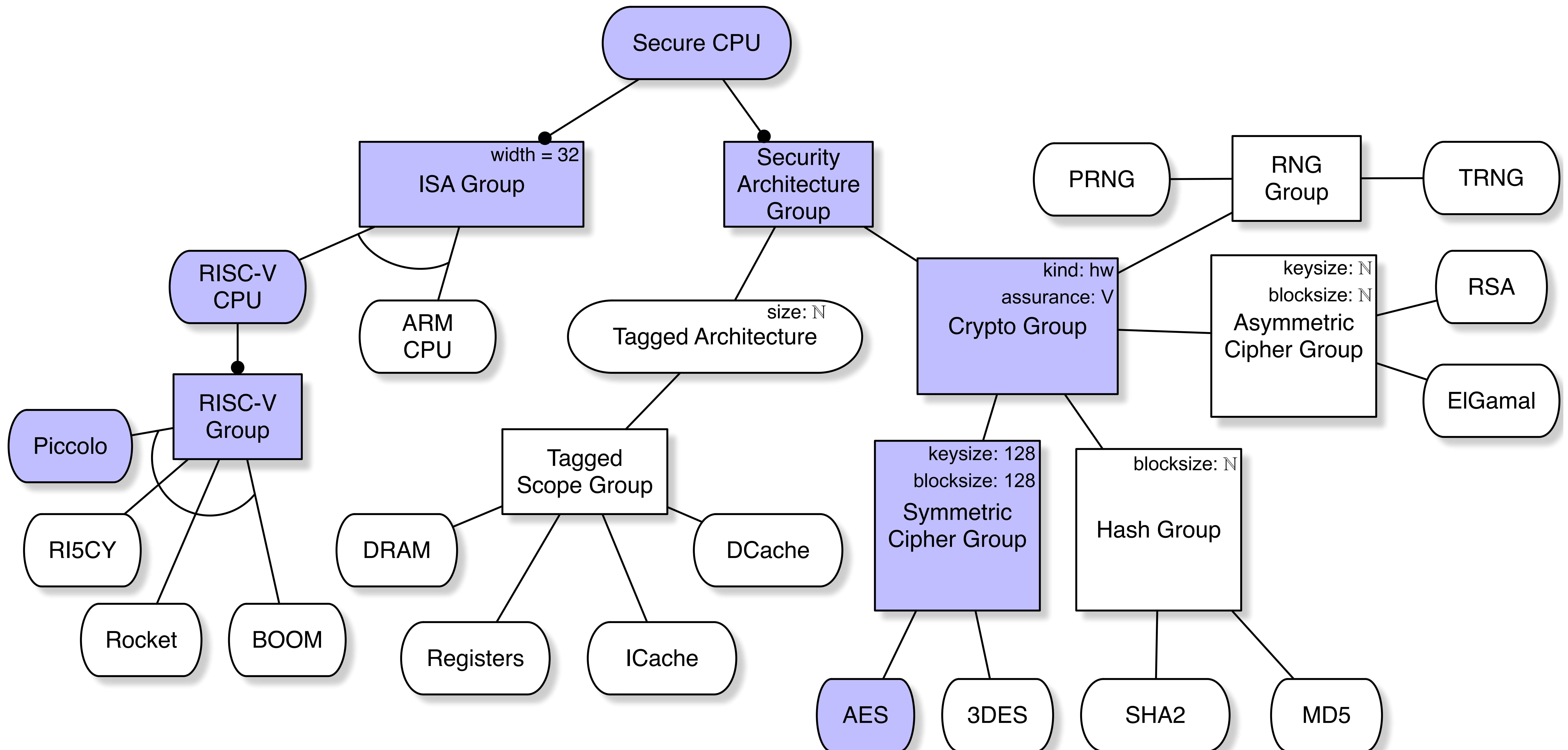
© 2018 Galois, Inc.

Example Feature Model (Unconfigured)



© 2018 Galois, Inc.

Example Feature Model (Configured)



© 2018 Galois, Inc.

R&D Status

- **LANDO DSL**: early stages of development — we expect an initial version before the 2018 RISC-V Summit
- **security test suite**: early stages of development
- **metrics and measures**:
  - for PPA, currently evaluating existing open and commercial measurement tools
  - for security, currently synthesizing/extending existing metrics work from NIST, MITRE, others
- **dashboard**: early stages of design, based on our previous work on software engineering dashboards
- **feature model generation**: early stages of design

© 2018 Galois, Inc.

Open Challenges

- accuracy of measures
- evolution of security metrics
- addition of more commercial tools on the backend for validation, verification, and measurement

© 2018 Galois, Inc.