|galois|

High Assurance Rigorous Digital Engineering for Nuclear Safety (HARDENS)

Joe Kiniry, Galois (kiniry@galois.com)

May 2023

Theme: Driving FM to Practice

Keywords: digital engineering, model-based engineering, software engineering, hardware engineering, safety engineering, requirements engineering, formal verification, rigorous runtime verification, Cryptol, SAW, ACSL, SysML, FRET, RISC-V

This work was supported by the U.S. Nuclear Regulatory Commission (NRC), Office of Nuclear Regulatory Research, under contract/order number 31310021C0014.

All material is considered in development and not a finalized product.

Any opinions, findings, conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the NRC.

galois

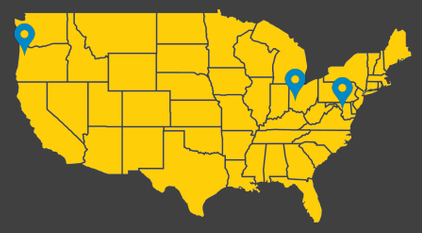
Advancing computer science R&D
Creating trustworthiness in critical systems

Clients

DOE
AFRL
DHS
DARPA
NSA
AMAZON
and many others
NASA
SDA
NIST

Offices

Portland, OR
Dayton, OH
Arlington, VA
Minneapolis, MN



History

Founded in 1999
100+ employees

A different kind of company

No managers

We have no fixed hierarchy of rigid positions and titles, and no traditional managers.

Radical transparency

Everything is transparent by default: company financials, decision making, open meetings, and even salaries.

Choose your work

Research engineers choose the projects they work on, and move freely between projects depending on personal interests and career goals.

Ownership

Employees own the majority of the company together, making important decisions as a group and partaking in the financial success of the company.

More info at lifeatgalois.com

Outline

- What is Rigorous Digital Engineering (RDE)?
- The HARDENS Case Study
- Where is RDE going next?

What is Rigorous Digital Engineering (RDE)?

The RDE Research Program

- **Rigorous Digital Engineering** (**RDE**)
  - **Rigorous** = use *applied formal methods* to reason about *models, implementations, and evidence*
  - **Digital** = use digital, mechanized, computational, denotational and executable models of components and systems to create *digital twins* (executable digital representations of components and systems) and *digital threads* (relations between digital and physical artifacts with known, evidence-based fidelity)
- **Engineering** = process, methodologies, tools, and technologies supporting *all* forms of engineering (particularly domain, requirements, product line, software, firmware, hardware, safety, systems, reverse, and security engineering)

Impact

- adopting RDE ideas typically…
  - decreases effort, and thus cost
  - shortens POPs
  - changes focus and shifts thinking “left” and “up”
  - significantly decreases the volume and changes the nature of bugs found during development
  - increases assurance and changes its nature
  - helps certification, risk analysis, and transition

The Technologies of RDE

The technology stacks supported thus far by the RDE methodology include:

- many different kinds of programming languages (procedural, object-oriented, functional, hardware, logic, and mixed-model, such as C, C++, C#, Rust, Haskell, Java, Scala, Kotlin, Eiffel, Chisel, Bluespec SystemVerilog, System Verilog, VHDL),
- specification and modeling languages (such as F*, ACSL, JML, CodeContracts, Alloy, Z, VDM, Event-B, RAISE),
- architecture specification tools and languages (such as Cameo, Rhapsody, MagicDraw, OSATE, Visual Paradigm and UML, AADL, and SysML, resp.),
- integrated development environments (such as Eclipse, Visual Studio, VS Code, IDEA, Eagle, KiCAD, mbeddr, Rodin, OSATE, Crescendo, etc.),
- formal modeling and reasoning tools (such as Alloy, PVS, Coq, Isabelle, UPPAAL, CZT, Overture, Rodin, Frama-C, SAW, Ivy, TLA Toolbox, FDR4, NuSMV, BLAST, and SPIN),
- operating systems (RTOSs, UNIX variants, seL4, etc.), and
- spans systems, hardware (ASIC and FPGA-based), firmware, and software

RDE at Galois

- RDE is used at Galois across dozens of Government and industry projects
- RDE is used, and was used previously, in several other companies and in the university R&D setting
- RDE permits Galois to create high TRL level systems with the following qualities
  - high-assurance for correctness and security,
  - demonstrable, open, peer-reviewed evidence, and
  - when compared to industry baselines for effort, cost, duration, risk, and outcomes, we find that we are virtually always lower effort, cost, duration, and risk, and yet produce much higher quality outcomes

RDE is not just for Galois

- I have taught others how to do RDE for decades
- the youngest students were sophomores at various universities in software engineering courses
- the most experienced students are full professors and senior Government R&D staff
- DARPA recently funded Galois to develop an updated RDE course (under the Space-BACN project), and we are/will be using RDE to develop that satellite platform, esp. its security subsystem
- we intend to give the course at Galois soon, recording the lectures and publicly sharing all course materials, project work, etc. with the world

The HARDENS Case Study

HARDENS

- **HARDENS** (*High Assurance Rigorous Digital Engineering for Nuclear Safety*) is a R&D project run by Galois for the *Nuclear Regulatory Commission* (NRC)
- the purpose of HARDENS is to demonstrate and educate about cutting-edge, high-assurance model-driven engineering
  - our focus is on nationally critical infrastructure, and thus safety-critical embedded systems
- within HARDENS, Galois has designed and built a demonstration Reactor Trip System (RTS) that is representative of a Digital Instrumentation & Control (DI&C) system for a Nuclear Power Plant (NPP)
  - the RTS is fault-tolerant and high-assurance
  - the RTS has a physical manifestation (an FPGA board plus sensors/actuators) and a set of digital twins

What did HARDENS Demonstrate?

- HARDENS demonstrates the creation of a complex system using a specification-driven approach.
  - both Design-by-Contract and Correct-by-Construction approaches are used in the project
- HARDENS demonstrates the generation of correct-by-construction evidence in the process of developing a reactor protection system, using well-known but rarely applied software engineering principles
- HARDENS also demonstrates, in our experience, the most economic approach to achieve the level of assurance needed for systems of such criticality — it shatters the myth that high quality entails high development cost

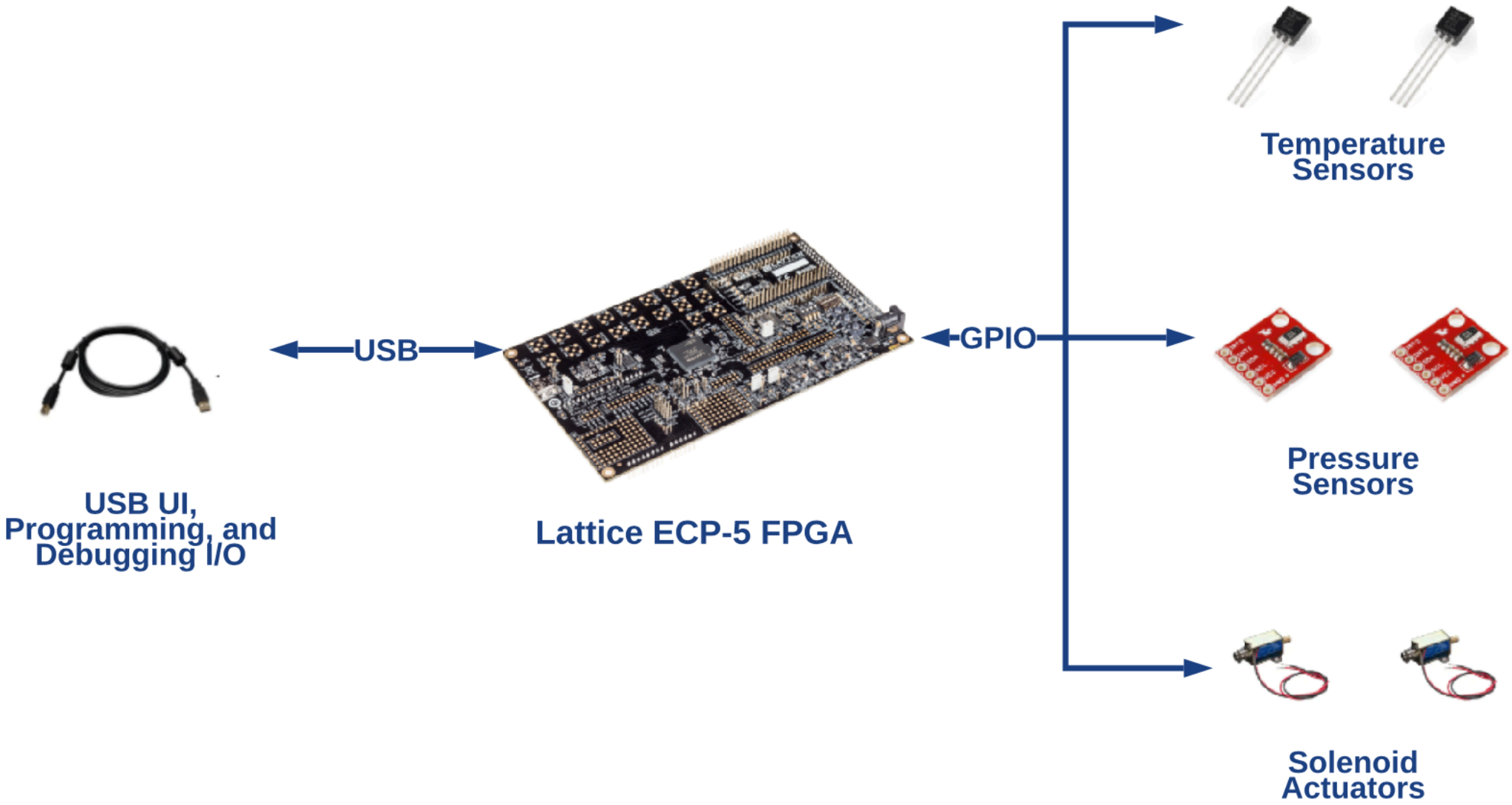
The Reactor Trip System (RTS)

- a seemingly simple control system that measures temperature and pressure of a reaction vessel and actuates solenoids to open and close valves
- representative of a class of systems that are deemed nationally critical infrastructure by the DHS
- a fault-tolerant system with *heterogenous redundancy*
- must fulfill a set of assurance “characteristics” found in the IEEE Standard 603-2018 “IEEE Standard Criteria for Safety Systems for Nuclear Power Generating Stations”

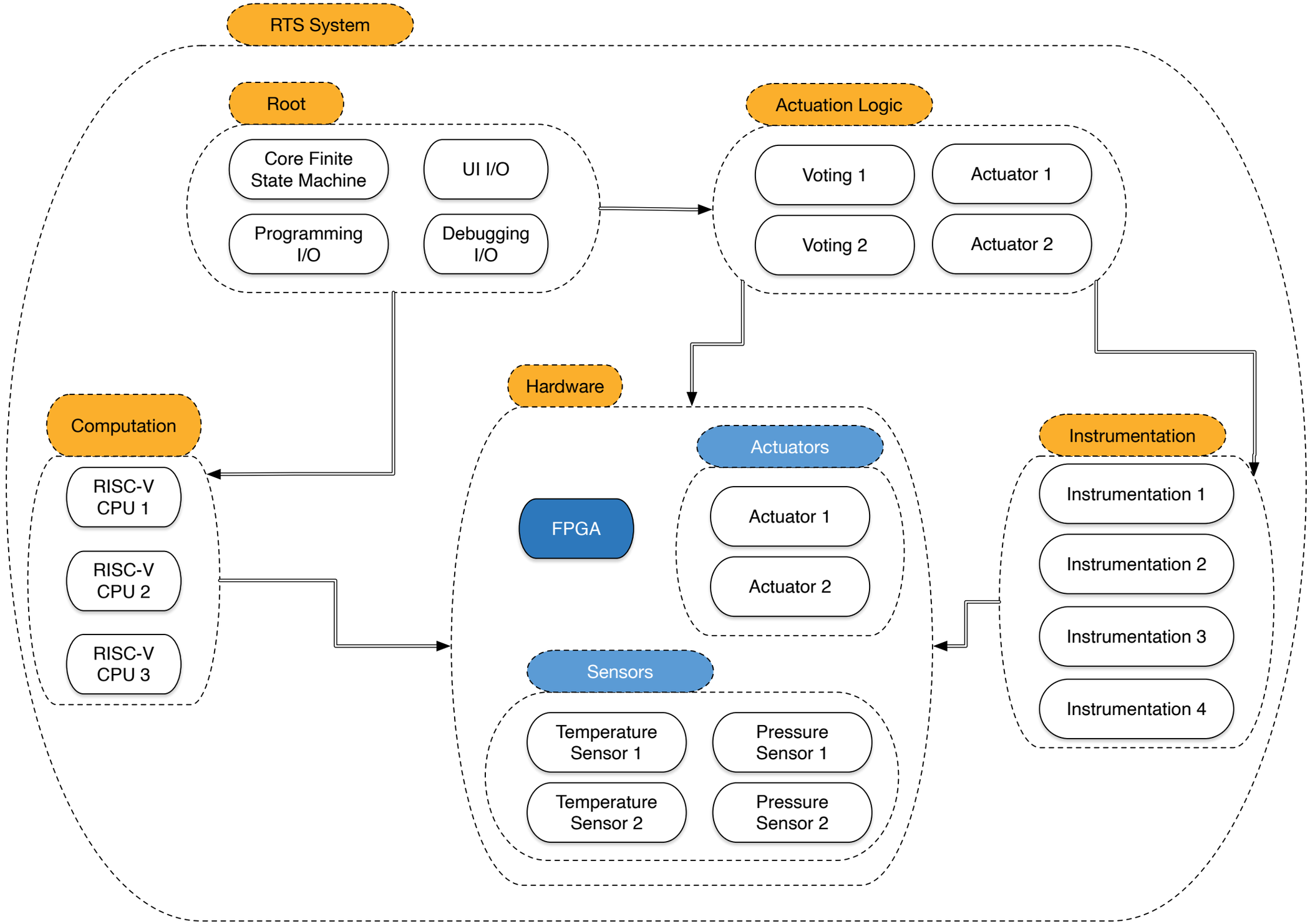
Digital Instrumentation & Control

- The RTS is a demonstration DI&C system, which…
  - are basically sense-compute-control architectures that control safety-critical systems,
  - have human-in-the-loop user interfaces,
  - commonly have built-in self-test subsystems that must be able to self-test/assure a system *while it is in operation,*
  - are fault-tolerant and have heterogenous components, where components are implemented using multiple techniques, sometimes by multiple teams, and have multiple redundant connectors,
  - are built today within the NPP industry without software or COTS hardware, and
  - the NRC does not want to see the industry repeat the technology mistakes of other nationally critical infrastructure industries.

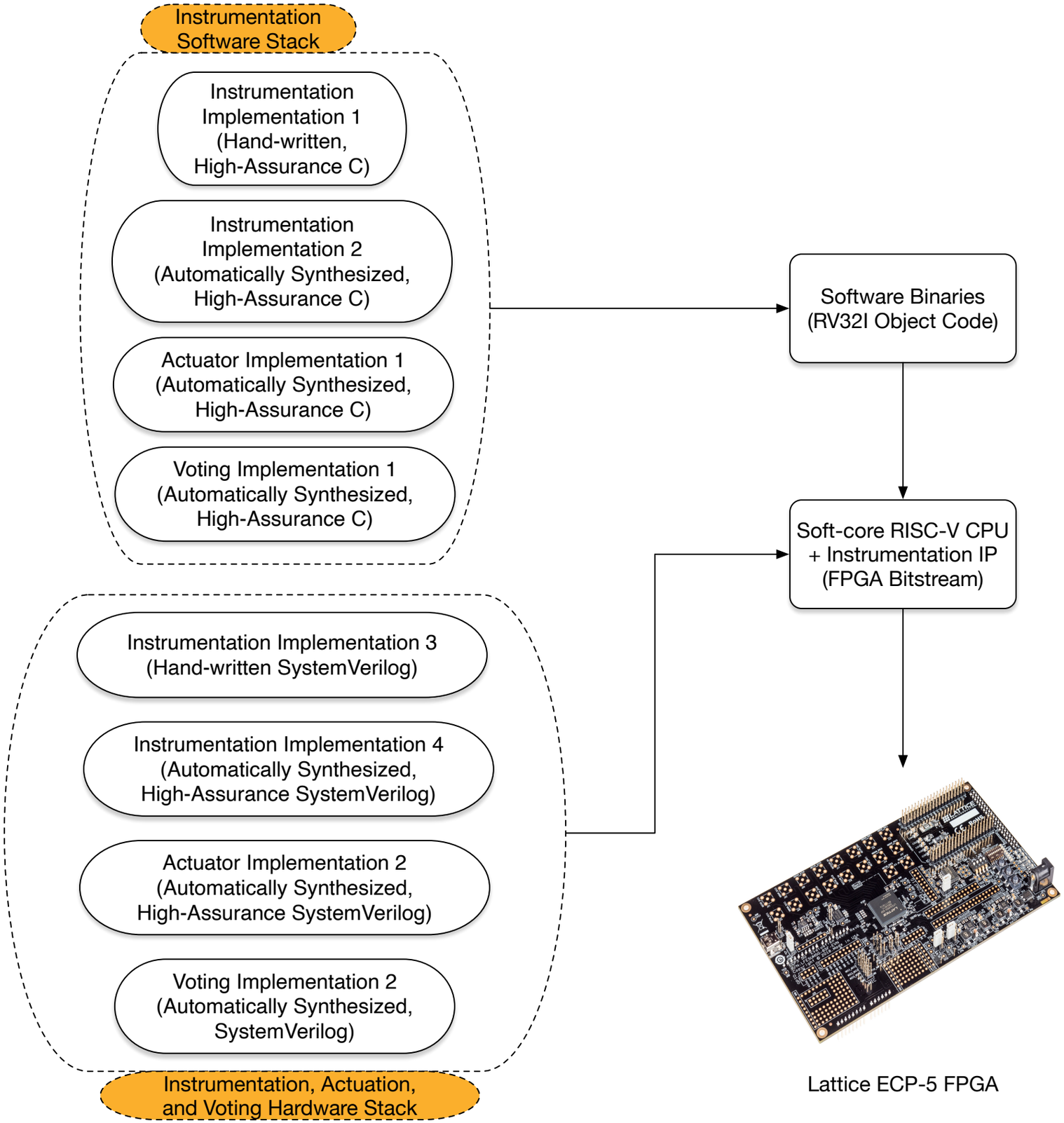
The RTS Hardware Artifacts



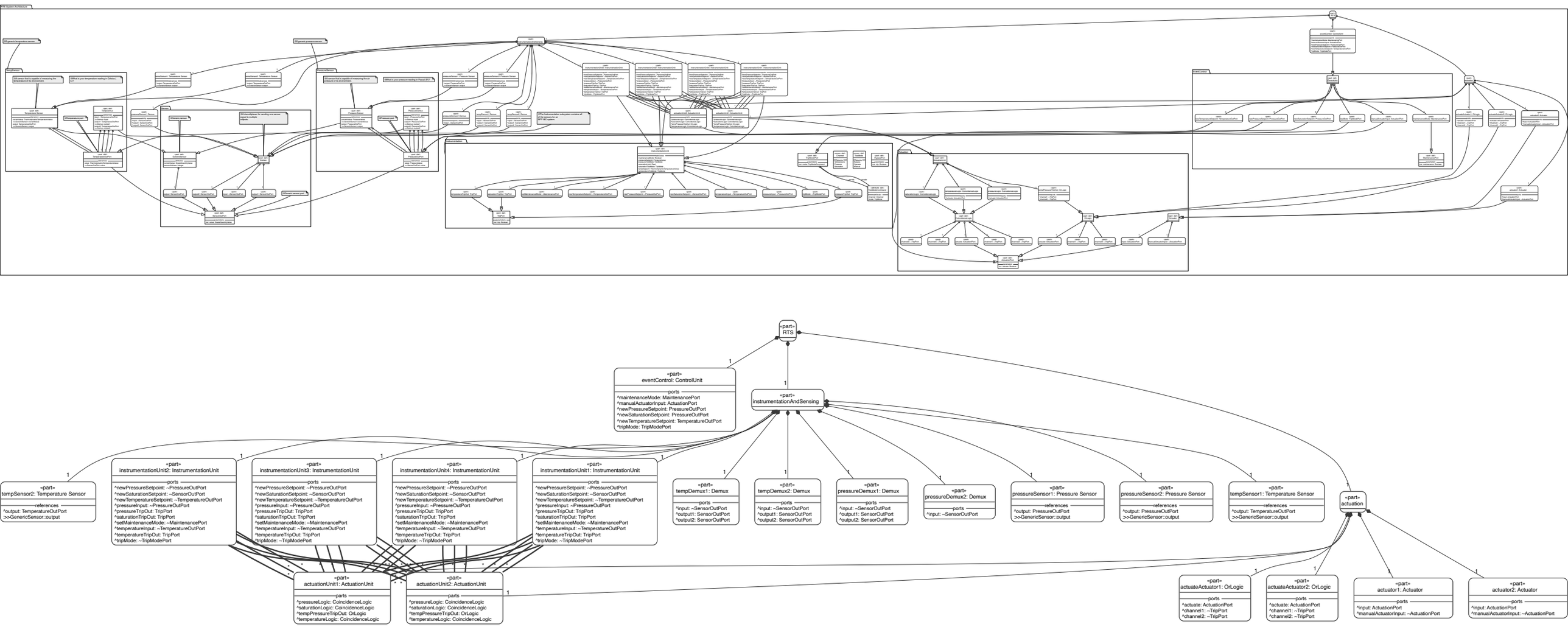
System Architecture



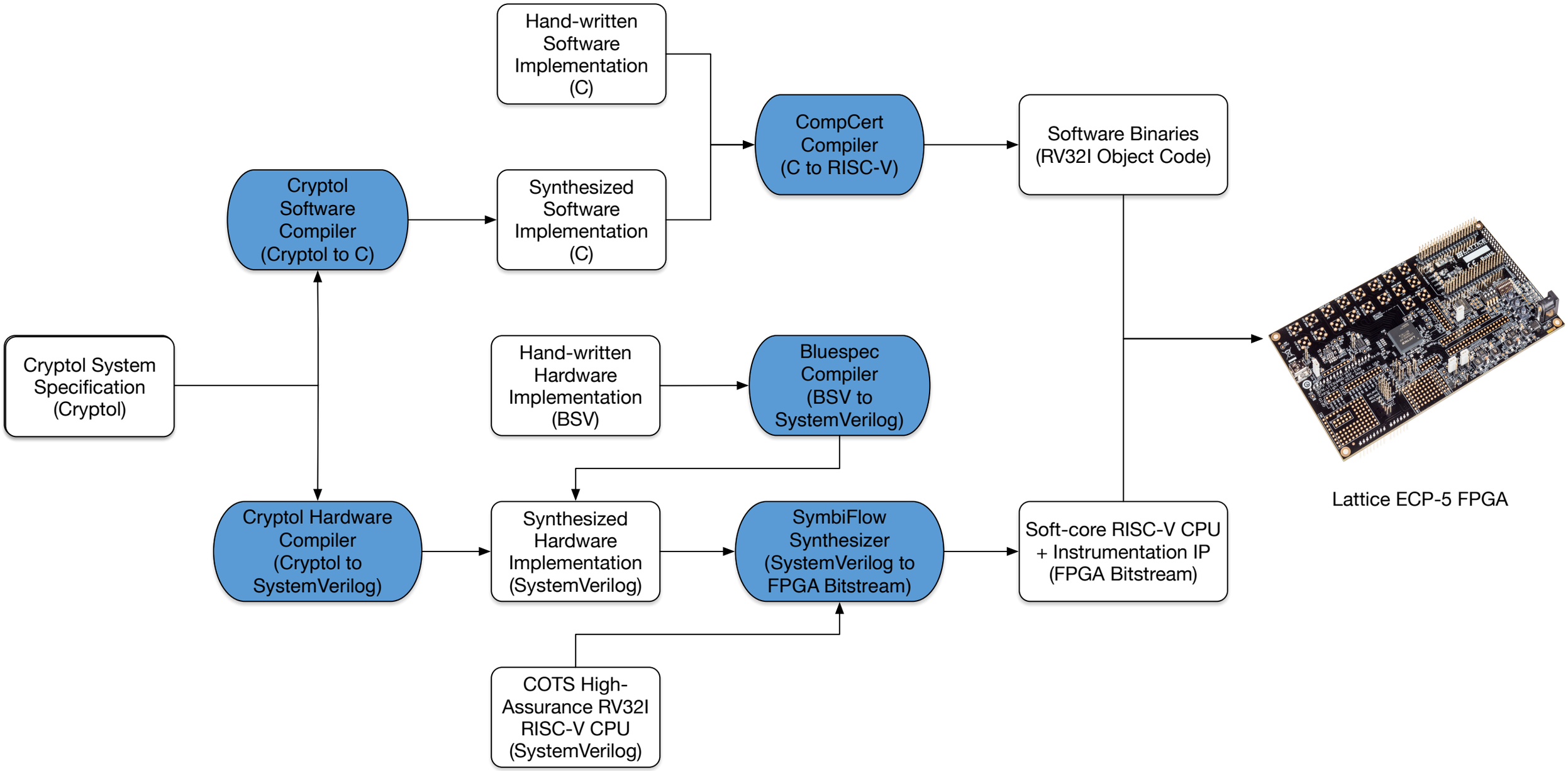
RTS Instrumentation Architecture



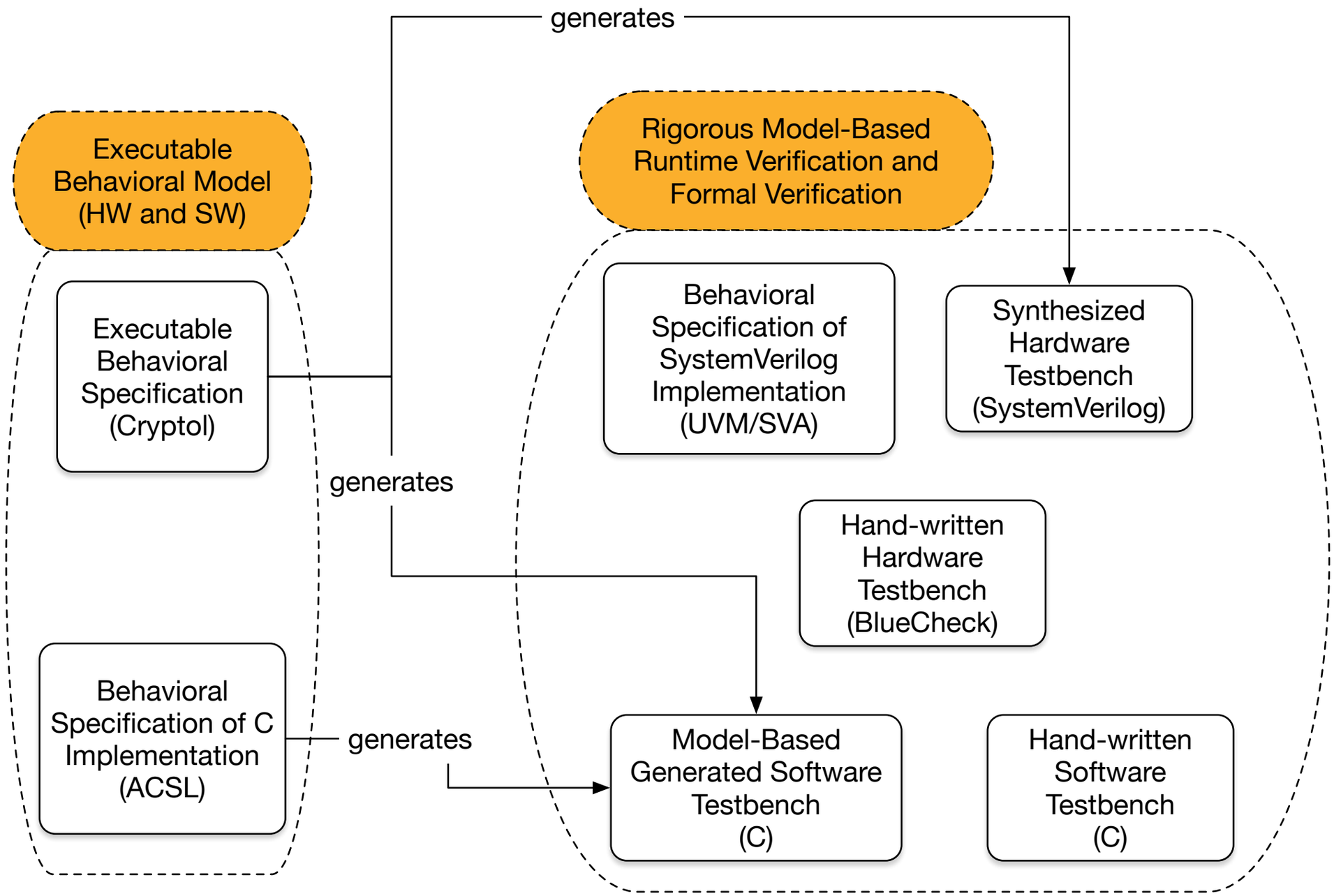
RT System Architecture in SysMLv2



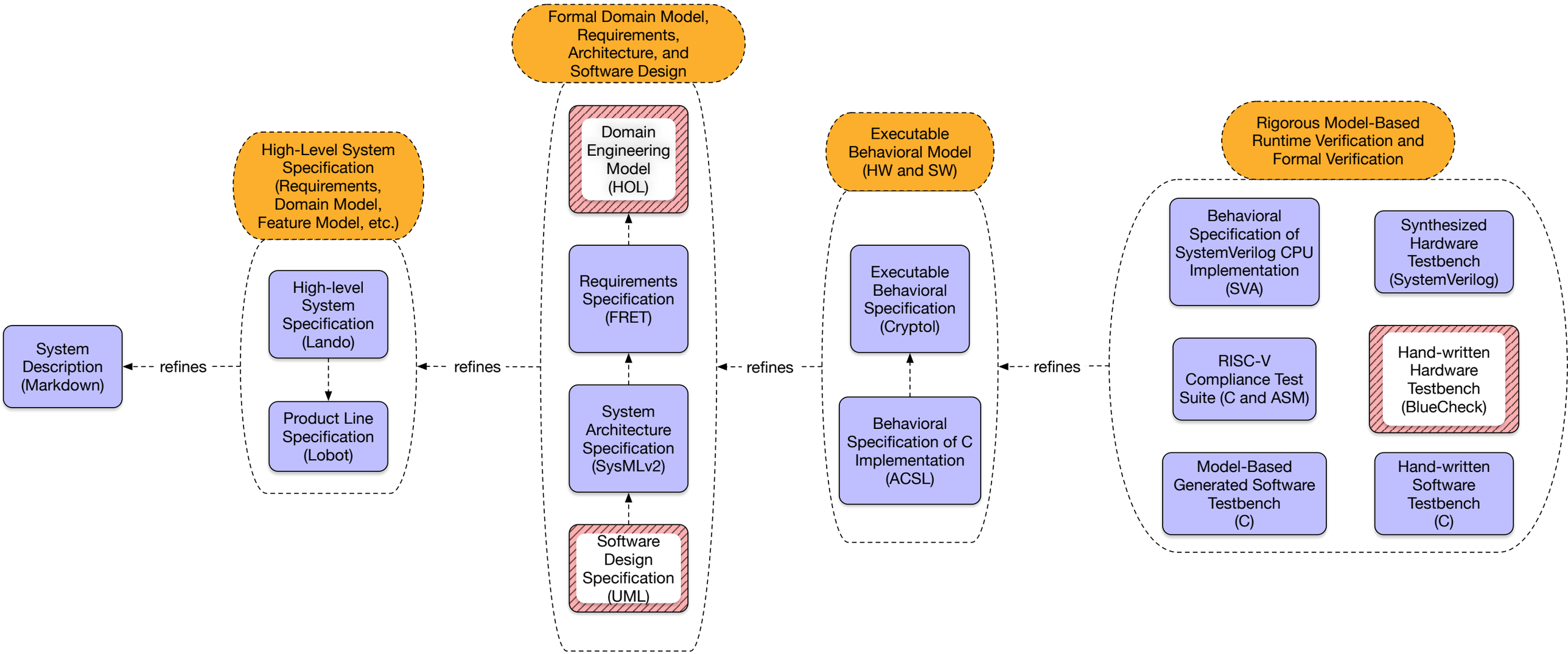
Implementation Artifacts and Relations



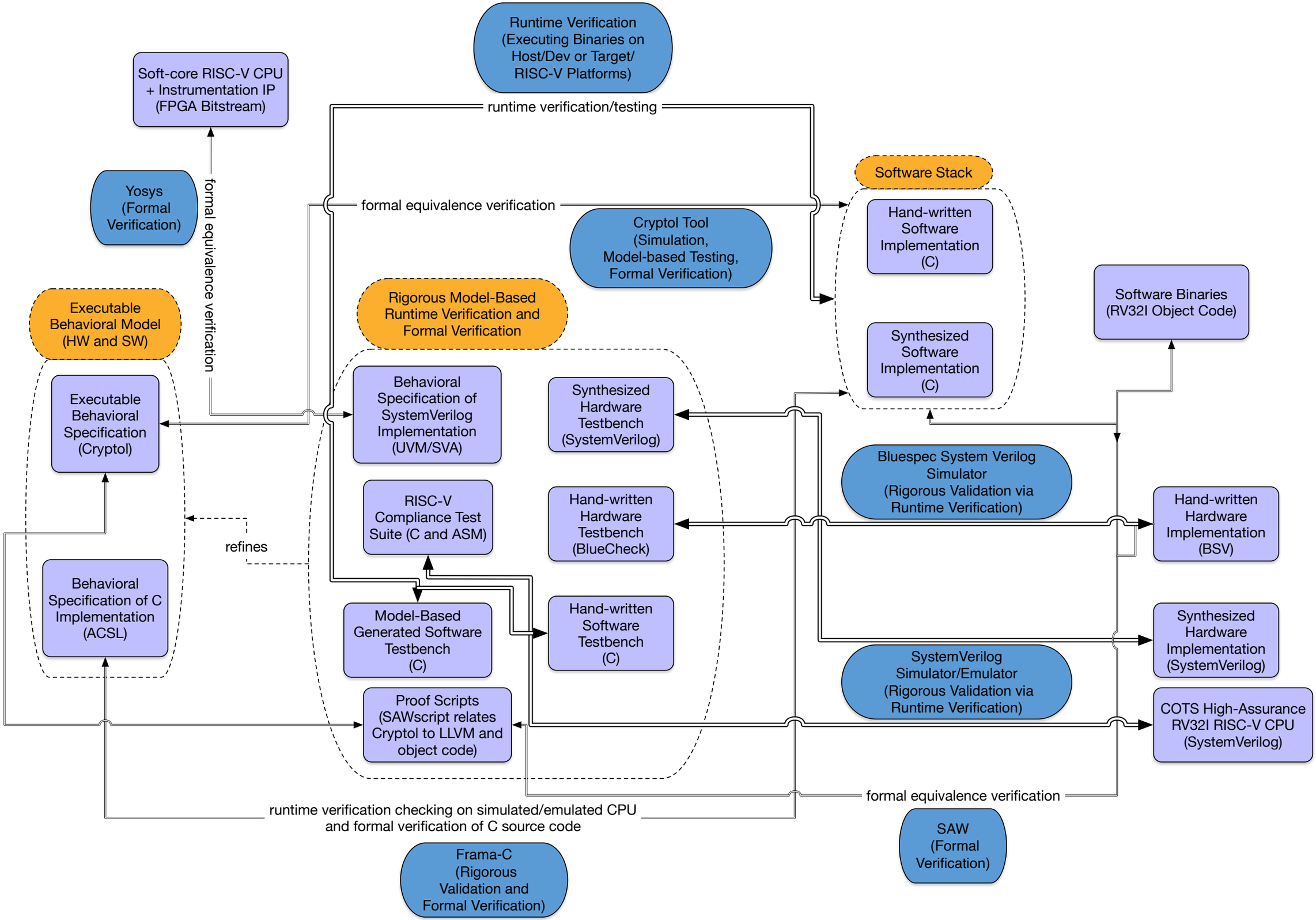
Generation of Assurance Artifacts



RTS Specification Artifacts



RTS Validation/Testing and Verification Artifacts



Kinds of Reasoning

- one can reason about a thing (a model, a software implementation in source code, a binary, a hardware design at various abstraction levels, etc.) in many different ways
  - model check states
  - attempt to satisfy a proposition
  - interactively prove theorems
  - our tools automatically generate theorems about artifacts by statically translating the artifacts into their semantic representations,
  - embed that translation in a background theory of the universe of discourse (using, in part, the domain engineering model),
  - and then state and try to prove theorems or generate other non-proof evidence about the relationship between artifacts

Equivalence and Refinement

- The main relationships that we specify and reason about in our RDE are equivalence and refinement.
  - *Equivalence* means that two artifacts are “equal” under some formal equivalence relation defined in terms of the underlying semantics used to describe the artifacts.
    - Cryptol spec ≈ C program
    - C program ≈ Java program
    - LLVM bitcode ≈ Verilog spec
  - Equivalence facilities reasoning about correctness across models, languages, optimizations, evolution.
- *Refinement* means that one artifacts “refines” another under some formal refinement relation defined in terms of the underlying semantics used to describe the artifacts.
  - C ⫤ ACSL ⫤ Cryptol ⫤ FRET ⫤ SysMLv2 ⫤ Lando ⫤ Markdown
  - SystemVerilog ⫤ SVA ⫤ Cryptol ⫤ FRET ⫤ SysMLv2 ⫤ etc.

Traceability as Refinement

- traceability in our formal method is effected through refinement and is *implicit* and *discoverable*
- when defined properly, refinement relations enable…
  - refinement checking
    - “Is A a refinement of B, and if so, how, and if not, why?”
  - model lifting
    - “What is a formal model of this artifact?”
  - model or code generation
    - “What is an artifact that refines this artifact?”

Explicit and Implicit Traceability

- *explicit* annotations for traceability are only used to connect development artifacts to process
  - e.g., annotation of a design decision on a specification that links to a GitLab issue that provide evidence and provenance and connects to a specific signed pull request and release tags
- nearly all traceability in RDE is implicit, and grounded in the use of consistent naming and structuring captured by the domain engineering model and the methodological decisions of a team
  - the *RDE Refinement Finder* tool automatically discovers implicit traceability and generates links for bi-directional equivalence and refinement relations

Assurance Case

- for high-assurance systems created with rigorous digital engineering, we must demonstrate that…
  - the specification which describes the system is the right specification (it is consistent, realizable, and has all of the properties we demand—aka spec validation)
  - all specifications relate to each other as specified to build the compositional argument of the system's properties (equivalent specs are equivalent, specs that are refinements are refinements, etc.)
  - the implemented system conforms to its specifications (it is shaped like, and operates as, promised—no more, no less—in precisely characterized environments)
- for HARDENS, in order to build this compositional assurance case we use the Lando type checker, the SysMLv2 type checker, FRET, Cryptol, SAW, Frama-C, and Yosys

Dynamic Assurance Case

- dynamic assurance classically means “run hand-written tests on the real platform”
- in the DevSecOps universe, it typically means “run hand-written and some automated tests on a test platform that is meant to duplicate the real one”
- in the MBE universe, it means “run tests that are automatically generated from models on…”
- in the Digital Engineering universe, it means “run tests on Digital Twins and the real platform…”
- in the RDE universe, it means “run rigorously created (>>99% automatically generated and a bit of hand-written) parametrized property-based tests derived from theorems about the architecture and assurance goals on all digital twins and the real test and deployment platforms”

Dynamic Assurance for HARDENS

- all Markdown requirements refine to SysMLv2 requirements refine to Cryptol properties (first-order theorems about the behavioral model) refine to property-based tests about the hardware and software
- refinement from formal Cryptol properties results in…
  - theorems stated in ACSL about the software (at the unit, subsystem, and system level, & software device drivers),
  - theorems stated using Bluecheck, SVA, and UVM about the hardware (CPU & SoC & hardware device drivers), and
  - a software testbench is automatically generated from the Cryptol model by translating every theorem into a software property that is runtime or formally verified

Dynamic Assurance for HARDENS

- all tests are design to run on…
  - the executable behavioral Cryptol model,
  - a POSIX emulation of the RTS hardware,
  - a hardware simulator of the CPU to verify that the CPU conformed to the RISC-V ISA spec and the (single-core and multi-core) SoC to verify SoC-level properties about devices, I/O, concurrency, and
  - the deployment hardware (the Lattice FPGA—not demonstrated yet in HARDENS)
- tests are run with RTS self-test enabled and disabled
- the entire test system is specified with a product line engineering *feature model,* and we aim to runtime verify all realizable models (e.g., identical results across three C compilers targeting three ISAs)

Model Validation

- to ensure that our formal models are valid…
  - we use FRET to formally analyze the system requirements for consistency, completeness, and realizability
    - FRET's realizability checker, which verifies that for a set of input values satisfying the requirements, a set of output values exists that satisfy the requirements
    - dually, the tool identifies inputs for which no output is feasible, which helps narrow down subsets of requirements that are inconsistent.
  - we use Cryptol to demonstrate (either test automatic property-based testing or, in the main, formally prove) that all theorems about the Cryptol behavioral model hold
  - we use Frama-C to demonstrate that all theorems about the software model are consistent by attempting to prove “bottom”
  - we use Yosys to demonstrate through formal verification that all theorems about the hardware hold by attempting to prove “bottom”—the implementation that crashes—satisfies the spec

Hardware Formal Verification

- our hardware has three kinds of components…
  - hardware device drivers for I/O
  - a 32-bit RISC-V CPU (NERV CPU from YosysHQ)
  - a single core and three core NERV-based SoC (the single core design is complete; the other is not)
- thus we must verify…
  - each device driver conforms to its formal model
  - the CPU exactly conforms to a specific flavor of RISC-V
  - the SOC behaves exactly as we specify

Hardware Formal Verification

- yosys is an open source tool suite that includes tools to synthesize bitstreams to some FGPAs and a formal verification tools that can perform equivalence checking & property-based verification
  - thus we use yosys to re-run the existing ISA formal verification on the CPU (as a kind of formal regression analysis), and
  - we runtime verify all SoC properties on the synthesized SoC (in simulation and on the FPGA)
- we use our correct-by-construction tools to automatically synthesize hardware implementations of some of the architecture from the Cryptol model

Software Formal Verification

- to formally verify the RTS software…
  - we use our correct-by-construction tools to generate some software components directly from the Cryptol model
    - then we use the SAW tool to further guarantee that the generated components are, in fact, correct through formal verification
- we use formal static verification of the C source code to
  - formally prove functional correctness relative to the Cryptol model, and
  - guarantee the absence of any runtime errors (e.g., due to array out-of-bounds errors and null pointers)

Digital Engineering Artifacts

- our RDE model includes system models that capture everything from domain engineering model and requirements (Lando and SysMLv2) to low-level architectural structure and properties (SysMLv2, Cryptol, ACSL, and SVA)
- we must demonstrate that our several digital twins are correct and are refinements of each other
- we must understand and quantify their fidelity, assurance level, and performance

Digital Twins

- the RTS system includes several digital twins
  - the executable behavioral Cryptol model,
  - a POSIX emulation of the RTS hardware,
  - a hardware simulator of the RISC-V CPU, and
  - the initial design of a (single-core and multi-core) SoC
- all system properties must be/are valid for all twins
- there are several other twins we would like to add
  - a machine emulator model (QEMU) of the platform
  - a Bluespec event-based simulator of the SoC
  - a SystemC-based platform emulator such as Imperas's
  - an alternative software-based hardware simulator such as Metrics's cloud/LLVM-based simulator

Digital Threads

- a digital thread is a chain of properties that run through refinements of digital twins and the deployment platform
- *traceability via refinement* guarantees that digital threads are *sound* and *complete*
  - from every relevant domain-specific concept in the domain engineering model to…
  - every system characteristic and requirement to…
    - every structural element and formal property of every model and digital twin…
    - to every structural element and formal property of the deployment hardware
- threads are *sound* when their elements (theorems/properties) are shown to hold for every twin and target in the refinement chain
- threads are *complete* when every abstract element is shown to hold for every twin and target in the refinement chain

Where is RDE going next?

Next Steps

- there are enormous opportunities in the design and implementation of SysMLv2 to facilitate RDE
- we need a formal, denotational, executable model of SysMLv2 to realize SysMLv2 digital twins directly and to define formal refinement relations to other formal models
  - we have a draft of a partial HOL model in PVS
- we need a means by which to validate and verify SysMLv2 implementations are conformant to its formal semantics
- we need an IDE that facilitates automatic refinement checking and refinement-based model lifting & generation
  - we need to define a refinement between SysMLv2 and AADL
- we need high-quality, interactive graphical model editing, refactoring, and rendering
  - preferably defined via an equivalence relation between the textual and graphical semantics, and then extracted to an implementation

More Information

- HARDENS Case Study https://github.com/GaloisInc/HARDENS (live on 1 June 2022)
- Galois https://galois.com/ & https://lifeatgalois.com/
- NRC https://www.nrc.gov/
- Cryptol https://cryptol.net/
- Lando https://github.com/GaloisInc/BESSPIN-Lando
- SAW https://saw.galois.com/
- Frama-C https://frama-c.com/
- ACSL https://frama-c.com/html/acsl.html
- FRET https://ti.arc.nasa.gov/tech/rse/research/fret/
- Yosys https://www.yosyshq.com/
- Coq https://coq.inria.fr/
- PVS https://pvs.csl.sri.com/
- Galois is hiring in this R&D area! https://galois.com/careers/