TECHNICAL LETTER REPORT
TLR-RES-RES/DE-2024-005

The HARDENS Final Report

May 2024

Authored by:
Joseph Kiniry, Alexander Bakst, Simon Hansen,
Michal Podhradsky, and Andrew Bivin
Galois

Contracting Officer Representative:
Derek Halverson
U.S. Nuclear Regulatory Commission

Division of Engineering
Office of Nuclear Regulatory Research
U.S. Nuclear Regulatory Commission
Washington, DC 20555–0001

Prepared as part of contract/order number 31310021C0014,
"Assessment of Model-Based Systems Engineering Processes in a Regulatory Review Context for Digital Instrumentation and Controls (I&C) of Existing NPPs"

This report does not contain or imply legally binding requirements. Nor does this report establish or modify any regulatory guidance or positions of the U.S. Nuclear Regulatory Commission and is not binding on the Commission.

FOREWORD

The following report was produced as part of work performed for a future focused research project for the US Nuclear Regulatory Commission titled "Assessment of Model Based Software Engineering Processes in a Regulatory Review Context for Digital I&C of Existing Nuclear Power Plants."

This research project contracted an experienced entity to implement a simple protection system using both: (1) highly integrated computer-based engineering development processes, and (2) model-based engineering. This would serve as a pilot that starts with a defined protection system, applies model based methods to its development, and would then support assessment of how the methods and their artifacts demonstrate technical adequacy against related regulatory criteria. The result was the High Assurance Rigorous Digital Engineering for Nuclear Safety (HARDENS) project, which is described in this report. It should be noted that the HARDENS project serves as a demonstration of concepts and methods, and was only required to implement portions of the functional and regulatory requirements of a reactor protection system. The demonstration of its technical soundness was to be at a level consistent with satisfaction of the current regulatory criteria, although with no explicit demonstration of how regulatory requirements are met.

The HARDENS project has continued beyond the duration of NRC funded research. The HARDENS code, as well as presentations, updates on this report, and other documentation are located on GitHub at: https://github.com/GaloisInc/HARDENS

Page intentionally left blank

HARDENS Final Report

Joseph Kiniry, Alexander Bakst, Simon Hansen, Michal Podhradsky, and Andrew Bivin

Galois

October 2022

Contents

Chapter 1

Executive Summary

This is the final report of the *High Assurance Rigorous Digital Engineering for Nuclear Safety* (**HARDENS**) project. In this project, Galois has developed a high-assurance, safety-critical demonstration system for the Nuclear Regulatory Commission using Rigorous Digital Engineering (RDE). The system in question is a Digital Instrumentation and Control (DI&C) system for Nuclear Power Plants (NPPs), and is called the Reactor Trip System (RTS).

RDE is the combination of *Model-based Engineering*, *Digital Engineering*, and *Applied Formal Methods*. The engineering focus of RDE is broad, as we have used it to perform *software, firmware, hardware, systems, domain, requirements, product line, safety, and security engineering* of *high-assurance, secure-by-design systems*. The HARDENS project includes nearly all of these kinds of engineering, but for security engineering at this time.

To our knowledge, this demonstrator is the most rigorously specified and assured system of its kind that includes formally assured software and hardware for a safety-critical system.

Model-based Engineering focuses on the use of semi-formal and formal models and their properties to describe aspects of a system independent of a particular implementation. Models are connected to each other through a variety of relations (refinement, containment, subtyping/subsumption/implication, traceability, etc.), models are either or both denotational or operational (and thus executable), and models are used to specify, examine, understand, and reason about a system well-prior to a line of code ever being written. Models are used for rigorous validation—through automatic model-based test bench generation and bisimulation—and formal verification—through automatic model-based verification bench generation.

The models used in HARDENS include, from most to least abstract:

- a Lando high-level system specification model, which includes within it:
  - a domain engineering model,
  - a requirements engineering model, which includes:

        * derived certification requirements,
        * contractual requirements,
        * safety requirements, and
        * correctness requirements.
    - a product line (feature) model,
    - a static system model,
    - a dataflow model of the RDE methodology,
    - a system event model,
    - a system scenario model (including all normal and exceptional behaviors),
    - a hardware, software, and evidence Bill of Materials (BOMs),

- a SysMLv2 system model, which includes within it, as refined directly from the Lando model:
    - a stakeholder model,
    - a domain engineering model,
    - a requirements engineering model,
    - property specifications for all correctness and safety properties derived from the formal requirements model,
    - a product line (feature/variant) model,
    - a static system model that includes both the software and hardware manifestations of the system,
    - a system action model, and
    - a validation and verification assurance case model.

- a formal requirements model expressed in JPL's FRET tool, as refined from the Lando and SysMLv2 requirements models above,

- a Cryptol model of the entire system, including all subsystems and components, including formal, executable digital twin models of the system's sensors, actuators, and compute infrastructure, and this Cryptol model includes a refinement of all formal requirements from FRET into Cryptol properties (theorems) about the Cryptol model itself,

- a model of the semantics of the RISC-V instruction set,

- a model-based specification of critical portions of the RTS's software stack expressed in ACSL,

- an executable and synthesizable Bluespec System Verilog model of a family of RISC-V-based SoCs, and

- a System Verilog executable and synthesizable model of a simple, in-order 32-bit RISC-V CPU (NERV, from YosysHQ).

Digital Engineering focuses on the use of *digital twins* of physical systems, subsystems, and their components. A digital twin is typically an executable model that has known and measurable objective fidelity in relation to the models or systems that relate to the twin. For example, an executable Cryptol or SCADE model are a digital twins.

The HARDENS system includes several digital twins, including simulation and emulation of the system hardware (CPUs and SoCs), software implementation, and system model.

Applied formal methods are the sensible use of formal methods concepts, tools, and technologies to formally specifying and reasoning about systems and their properties. In the context of RDE and the HARDENS project, we use formal methods to achieve the following assurance:

- critical components of the RTS are automatically synthesized from Cryptol model into both formally verifiable C implementations and formally verifiable System Verilog implementations,
- automatically generated C code and hand-written implementations of the same models are used to fulfill safety-critical redundancy and fault-tolerance requirements, and all of those implementations are formally verified both against their model, as well as verified against each other as being equivalent, using Frama-C and Galois's SAW tool,
- the RISC-V CPU is formally verified against the RISC-V ISA specification using the Yosys open source verification tool,
- the RISC-V-based SoC is rigorously assured against the automatically generated end-to-end test bench,
- the formal requirements specified in FRET are formally verified for consistency, completeness, and realizability using SAT and SMT solvers,
- the refinement of these requirements into Cryptol properties are used as model validation theorems to rigorously check and formally verifying that the Cryptol model conforms to the requirements,
- the Cryptol model is used to automatically generate a component-level and end-to-end test bench (in C) for the entire system, and that test bench is executed on all digital twins and (soon) the full hardware implementation as well, and
- all models and assurance artifacts are traceable and sit in a semi-formal refinement hierarchy that spans semi-formal system specification written in precise natural language all of the way down for formally assured source code (in verifiable C), (a side-effect of the optional use of CompCert) binaries, and hardware designs (in System Verilog and Bluespec System Verilog).

While there are numerous relevant artifacts contained in this report and in the project repository, this project's focus was not on correct-by-construction techniques nor on the assurance case for a model-based system. Such work is likely the focus of future, follow-up projects.

The appendices of this report are automatically generated from the project artifacts using the *RDE Refinement Finder* tool, developed in parallel with this project. As such, a large fraction of refinements embodied in the RDE process and methodology are discovered and hyperlinked therein, permitting one to navigate the project specifications, implementations, and assurance evidence via a PDF.

Clearly there is a lot more that such a tool can accomplish with further revision. The tool highlights when the RTS artifacts themselves are not in full alignment via refinement, so as this system sees future development and evolution, it will undoubtedly drift decidedly toward complete refinement via RDE.

Chapter 2

Introduction

Models have been a part of computer science since its very beginnings. Naturally, given that computer science is the daughter of mathematics, many early models were defined with foundational mathematics and logics, such as set theory and type theory.

Such models, while useful for describing and analyzing conceptual artifacts, are very often far from the day to day artifacts of engineering, such as the source code of an application or the design of a printed circuit board. Thus modeling languished into narrow subdisciplines of computer science, mainly those called "formal methods" and "programming languages."

Other disciplines of engineering, such as structural and mechanical, needed their models to be closer to reality decades before it was necessary in computer science. Partly this timing was driven by the real world requirements of the field: a bridge must withstand normal and extraordingary environmental forces on it for its lifespan. Partly it was driven by the cost of early mistakes and failures: correcting a building mistake is very costly in materials and time. Partly it was driven by matters of scale: e.g., the need to manufacture thousands of identical widgets as a part of a production line.

But mostly it was due to matters driven by external societal requirements due to the safety-centric nature of engineered structures, such as liability insurance, contractual culpability in the case of structural failure, and, indirectly, manifested in the form of engineering certifications.

Unlike much of software engineering, structural engineers do not get to "wing it" while building that bridge, changing their mind as they go because of earlier mistakes made in a rush. If the bridge collapses, they do not just shrug and try something again. It is not virtuous to define a team's goals every week and rush toward those goals, demonstrating that that frame of a house somehow resembles what the concrete structure will someday look like.

Over the past twenty-five years, models in computer science—particularly in software and hardware engineering—have crept closer and closer to the found, reused, and created artifacts of engineering. In the 1980s, models resembled code only if one squinted right through a very abstract lens; in 2022, models

and code are sometimes interchangable and deeply connected.

2.1 Early Modeling Languages: UML and SysML

The rise of popularity of object-oriented languages in the early 1990s was a catalyst for the rapid creation of, experimentation with, and standardization of a host of modeling languages, such as the Unified Modeling Language (UML) and its ancestors and predecessors. While millions of models were described with UML, rarely would two model readers or writers precisely agree upon what their shared model meant.

The fundamental reason why there are these kinds of disagreements is that UML was created *without a formal semantics*. Semantics give meaning to things—models and code—and without semantics, everyone, and every tool, is free to interpret the meaning of a concept in their own way. UML did have a *specification*, but it was written in English and was full of omissions, ambiguities, and contradictions.

In the early 00s, researchers from several institutions spent years formalizing parts of UML, but in the end there was no agreement upon what constituted **the** semanatics for UML, and few-to-no tools paid attention to semantics-centric research. This lack of formal foundation would come back to haunt not only those that tried to apply UML to rigorous software engineering, but also the future of systems engineering.

In the early 00s, development on the System Modeling Language, SysML, began. SysML was created as a *profile* on top of UML, thus it could both extend and leverage UML in a variety of ways: principly to design SysML's meta-model, to include a subset of UML diagrams about which there was already substantial use and practical agreement of utility. Unfortunately, because UML's lack of widely agreed upon semantic underpinnings, SysML took UML's albetross and fattened it up.

Consequently, SysML (version 1, hereafter written as *SysMLv1*), while widely deployed now within industry and the DIB, is mainly used to specify the simplest and least semantic of system properties: static architectures, structured system requirements, and a small subset of behaviors at best.

Thus, in the end, UML and SysMLv1 are widely used, but are not "meaningful" in a formal sense, since they do not have accepted semantics that are concretized in tools, and have few means by which to connect models and code, as discussed later in section 2.4.

2.2 Semantic Modeling

In order to have "meaningful" models, models must be written in languages that have a formal description which is amenable to automated or interactive reasoning. A formal description typically must include: (1) a concrete and abstract syntax, (2) that syntax may be either or both textual and graphical, (3) a

type system which denotes which specifications are well-formed and meaningful, and (4) one or more semantics (axiomatic, operational, or denotational) in order give meaning to any specification.

Also, in order to be a *useful* modeling language, the language must also have (5) tools which support the efficient writing, rendering, type checking, and dynamic or static reasoning about specifications.

Many modeling languages have been invented over the decades that fulfill some of these requirements, and some of those modeling languages fulfill all of these requirements. Some of the more popular and high-impact high-level modeling languages that fulfill all of these requirements include (in alphabetical order): Alloy, B, BON, Event-B, Maude, OBJ and its daughters, SCADE (and its ancestor, Lustre), SPIN, UPPAAL, VDM and its daughters, Z and its daughters.

Formal specifications can also be written directly in logic. Logics come in various flavors, each with an increasing about of expressiveness that comes with an automation tradeoff: propositional logic is less expressive than first order logic (FOL), which is less expressive than second order logic (SOL), which is less expressive than higher order logic (HOL). The most popular tools that provide support for what amounts to "logical" modeling languages for these different logics include a variety of SAT (Boolean SATisfiability) solvers, Satisfiability Modulo Theory (SMT) solvers, FOL and SOL solvers, and Logical Frameworks (LFs) such as Coq, Isabelle, LEAN, and PVS.

Formal specifications written in these languages are necessarily "reasonable", as one must specify every single thing about a language for the tool to be 'ok' with its use. Of course, those languages that are only specified in an LF nearly always force one to interactively reason about specifications and their models, which is a complicated and sometimes expensive proposition at best. Languages that have *both* a complete HOL semantics *and* are supported by automated reasoning tools hit the sweet spot of well-foundedness, completeness, and usability.

2.3 Verification Competitions: Driving Tools to Utility

Formal modeling and reasoning tools are a dime a dozen. Over the years, hundreds of tools have been created to explore various specification and reasoning challenges and different domains. For a time, there was little coherence or coordination across the domain, thus to use two tools, one had to learn two input languages, and both tools' behaviors and capabilities.

Over time, this fragmentation was recognized as an impediment to transition and the joint/mutual use of multiple tools for a single problem or system. Thus, a consolidation of input formats began to occur.

Some input formats were determined by parties outside that of the modeling and reasoning community. If one is going to reason about programming language

X, your tool has to be able to read and understand X.

Other input formats were created with the express intent of having a shared language or platform across a subdomain of modeling or reasoning. Subcommunities of researchers came/come together to define modeling and specification languages by combining the best ideas of earlier work, and these shared languages provide the commonality necessary to enable multi-faceted modeling and reasoning.

Once this commonality came to pass, soon after contests associated with conferences emerged as a means by which to compare apples to apples in reasoning communities. Contests drive R&D teams in two ways. First, contests provide a means by which to challenge the community with new kinds of reasoning goals: goals of scale, new theories, new properties, etc. Second, contests drives teams to aggresively explore new algorithms, internal representations, programming techniques, and more in order to achieve community goals.

As a consequence, several different verification contests now regularly run spanning various branches of modeling and reasoning. For example, a few of the more prominent competitions include SV-COMP, MCC, SAT, SMT-COMP, and VerifyThis.[1]

SV-COMP is the "software verification" competition, which focuses on general-purpose tools for modeling and reasoning about software written in the C programming language. MCC is a competition that focuses on general-purpose model checking, mainly specified via a modeling formalism called Petri nets. SAT is the longest running competition, which focuses on problems in Boolean SATisfiability—a foundational abstraction used by many reasoning techniques. SMT-COMP complements SAT, insofar as it focuses on reasoning about the satisfiability of models, modulo various algebraic theories. Finally, VerifyThis is the big-picture, general-purpose competition that permits teams to use any approach the like to specifying and reasoning about models and implementations.

Modern advanced modeling and verification was enabled by four key environmental factors: (1) mathematical sophistication sufficient for real world program reasoning, reinforced by mechanized reasoning environments, (2) the researcher community's willingness and bravery to tackle real world programming and modeling languages, (3) the dramatic rise in computational resources (primarily the Gb boundary, as the sweet spot of productivitity hit once RAM sizes passed 1 Gb and CPU performance passed 1 GHz), but perhaps most importantly, (4) the competitive nature of these competitions, pushing research teams beyond the threshold of academic publication, and toward usability and robustness.

[1] There are many other competitions, many of which are summarized and enumerated on Alastair Reid's Verification Competitions blog post.

2.4 Grounded Modeling: Connecting Models and Code

While there are many kinds of modeling and reasoning, only a small fraction of modeling and reasoning is about *implementations*.

Partly this is due to the historic trends of research communities. Many, perhaps even most, lines of work in modeling and reasoning are classically grounded in the research area called *formal methods*.

Formal methods is the application of logical or mathematical techniques to the specification and reasoning of systems. Since formal methods is wholly grounded in formality via mathematics, it is no surprise that most of the early formal methods researchers were mathematicians. But for decades, "programming" was viewed akin to bricklaying, and thus was uninteresting to mathematicians-cum-computer scientists. Consequently, and perhaps surprisingly, the vast majority of modeling languages and their reasoning tools have little to nothing to do with *programs*.

Modeling languages and tools that have no direct connection to code are not inherently flawed, or the wrong tools to use. They can be used very effectively to reason about critical aspects of systems, especially systems whose models are terrifically complex, such as those that exhibit large amounts of concurrency. Likewise, thinking and reasoning at a more abstract level (at least, more abstract than raw code), or in a domain specific fashion, is commonly enormously more effective and scalable than reasoning about implementations in general purpose programming languages.

2.4.1 Disconnected Models

But in doing so, one must ask, "How is the analysis obtained by this language or tool affected in the system which it purports to describe?" More often than not, these kinds of "disconnected" models fall into two usage categories.

The first, and least authentic and useful, is what we will call "slideware." Models are created, rendered, and shown in documentation and presentations (thus, "slideware"), but they have little or nothing to do with the systems that are built. This usage scenario is typified by the creation of complex *static* models of systems—models that explain the parts of a system and how they interconnect, for example—but no *dynamic* or *behavioral* descriptions of the system, explaining what it does, or how it operates.

The second kind of commonly seen "disconnected" model is a model that is rich, authentic, and demonstrates aspects of a system, and yet has no strong connection, even informal, to the system implementation. This situation is commonly mitigated by putting into place a rigorous process and/or methodology that attends to ensuring that model elements and properties are traceable to system implementation elements and properties.

There are international standards, such as the ISO 90000 family of process standards and the ISO 27001 information security standard, that focus on

describing and evaluating rigorous processes. Likewise, there are certification regimes that mandate rigorous processes and methodologies, such as the FAA DO-178 series of standards for aircraft safety.

While such standards can be effectively used to maintain a connection between authentic models and implementations, doing so requires diligence and excellent deveopment and collaboration environment support for facilitating, reinforcing, and checking such human-in-the-loop processes. This pratice is also typically fairly resource intensive from a developer or compliance effort point of view.

2.4.2 Fully Connected Models

Fully connected models, on the other hand, are models whose relationship to implementations is direct and is checked by reasoning tools. Connected models come in three flavors.

Code Generated from Models. The most common fully connected model is witnessed in specification systems that include code generators.

Some code generators only turn model abstractions into code abstractions, such as UML-based code generators that convert class diagrams into type declarations, such as Java classes or interfaces or C++ header files.

Other code generators are capable of translating behavioral specifications of algorithms into fully functional implementations, such as the code generators for Cryptol and SCADE.

Finally, some logical frameworks are able to generate code, converting specifications, implementations, and proofs in logic into implementations in languages like OCaml, Haskell, Scala, and C.

In all three of these scenarios, untouched code generated from a model is meant to be in full, traceable alignment with that model. What "alignment" means is very much dependent upon the tool and the goals of code generation, ranging from "they look about the same" to "all properties of the formal model are guaranteed to hold for the generated code."

Models Abstracted From Code. A second kind of fully connected model is witnessed when models are abstrated from code. More generically, they are abstract models lifted from more concrete models or code, as the relationship need not be only between models and models.

Some tools that support UML support model lifting (in the form of class, object, sequence, and collaboration diagrams) from source code. Many of Galois's formal reasoning tools lift models from source code, binaries, or intermediate representations, such as LLVM.

These kind of fully connected models are the dual of the first case: untouched abstract models generated from more concrete models or code are meant to be in full, traceable alignment. And, as before, what "alignment" means varies considerably across technologies.

Models and Code in a Refinement Relation. A final kind of fully connected model-model or model-code relation is that of a refinement relation. *Refinement relations* are relations that preserve specific, well-specified sets of properties across the relation.

What kinds of properties are, or can be, preserved very much depends upon the refinement, and thus on the nature of the two sides of the relation. The idea of refinement is discussed a great deal more in later chapters.

2.5 Assurance Cases

Assurance cases describe a system, its purported properties, and the evidence that the system conforms to the properties. They are widely used in safety-critical and national defense domains, especially in sectors defined as nationally critical infrastructure.

A given assurance case focuses on a specific set of concerns. For example, one flavor of assurance case focuses on the *correctness* of a system, a second might focus on the *dependability* of a system, and a third might focus on the *safety* of a system.

Assurance cases commonly are stated and argued entirely informally. These kinds of "classical" cases are precisely written natural language documentation—say a large Microsoft Word document printed on paper which contains some figures, tabled, and a complementary Excel spreadsheet which explains traceability—that are read, reflected upon, and judged by subject matter experts.

It is not uncommon for such an informal assurance case to accompany a snapshot of part of the system it describes; say, for example, a printout of some of the system's source code. While assessors that are judging the quality and validity of the assurance case may, on occasion, examine those engineering artifacts, this is more the exception than the rule.

Only in the most rigorous safety-critical and security-centric certifications, such those of Common Criteria and the NSA, and their respective assurance cases are system engineering artifacts deconstructed and examined by experts carefully, often with some computational tool support.

2.5.1 Tool Support for Assurance Case Presentation

There are a broad range of assurance case presentation research tools that have been created over the past few decades. Each focuses on a different kind of structured assurance case, or is targeting a particular domain. In general, few are robust, scalable, and supported, and until recently, none included support for formal specifications and verification.

The recent ARCOS DARPA program program focused on this problem. It is meant to build a general-purpose assurance case database and reasoning system that can be used to help explain and vet the kinds of systems that the Department of Defense regularly buys and builds. ARCOS is not yet ready for

use on a system like HARDENS, but should be soon.[2]

Assurance cases use a variety of notations for presentation, such as Goal Structured Notation (GSN) and the Claims Argument Evidence (CAE) notation. Many tools focus on the idea that an assurance case is a model in and of itself, and one should be able to generate "classical" informal and semi-formal written artifacts directly from such a model.

Automation (re-)generation of evidence and reports is a boon to teams that must regularly go through a certification gauntlet, saving time and effort. And in combination with collaborative development tools, when used properly, can help track development, changes, artifacts, and evidence. After all, hand-writing assurance cases, even after reusing templates and boilerplate text, as we often see in evidence submitted with NIST FIPS-180 certification for example, is a very time-consuming an expensive proposition.

Such automation to check the process-centric boxes of the past is now, recently being complemented by automated, computational means by which to check and re-check assurance claims. Automatic claim checking is meant to be largely familiar to traditional software because it looks and feels like a test bench.

2.5.2 The HARDENS Assurance Case and its Presentation

The HARDENS project demonstrates this kind of automation in spades, using *Rigorous Digital Engineering* to state claims, build arguments, and demonstrate evidence. Evidence comes in several flavors, from structured, traceable design decisions to runtime verification using dynamic evaluation of models (also known as digital twins) and code to full formal verification using theorem proving.

Going into this project Galois wanted to explore using existing assurance case tools that have been developed by and for other agencies, such as NASA's AdvoCATE tool. But given the project's size and budget, we decided early on to simply present the assurance case directly in the project artifacts, and describe in this report how to read, understand, and evaluate those artifacts through example.

Several Eclipse-based frameworks (in addition to AdvoCATE), such as ACEdit, SafeEd, and AutoFocus3, have been created to implement fragments of GSN. Other open source or freely available tools that have novel features include CertWare, D-Case, EviCA, ACCESS, DiaSAR, the Evidential Tool Bus (ETB), and WEFACT.

Mature commercial are fewer in number, and include the widely used ASCE, Astah GSN, NOR-STA Assurance Case Tool, the Safety Case Toolkit (SCT), TurboAC, and Safety.Lab.

Denney and Pai [**DenneyPai18**] contains a good overview of the state of various commercial, research, and open source tools.

[2] Galois is a performer and evaluator on the ARCOS program, and is working with DIB partners on evaluation and transition of ARCOS's technologies.

If Galois continues to work with the NRC on formal assurance case development and presentation—especially with Rigorous Digital Engineering—we recommend that we jointly choose one of these existing tools as a foundation for future experimentation and use.

In addition to performing on the ARCOS program, at Galois we currently use the *Resolute* language and tool from Collins Aerospace on the DARPA CASE project. Its use is inappropriate for the HARDENS project as Resolute is a specification language concretizes as an annotation annex for the AADL system specification language. While we use AADL on a number of projects at Galois, given the size and budget for this project, we chose not to demonstrate it in the HARDENS demonstrator. AADL was mentioned in our proposal, and fits into system specification below SysML, as it is most often used to precisely specify an embedded system's structure and behavioral, especially in mission-critical settings.

Consequently, especially given the document-centric framing mandated by the NRC, we use a generic-but-rich means by which to state our model-based assurance case. The whole of the argument is grounded in the primary document that frames the project: the original NRC RFP. From that single source we derive all artifacts relevant to the system and its goals and properties, including international standards, research papers, relevant software and hardware, and more.

Chapter 3

Model-based Engineering

Several very good books have been written about model-based engineering (MBE) [2, 1, 3], though few focus on rigorous MBE that is grounded in the use of applied formal methods.

Model-Based Engineering (MBE) is fundamentally just engineering that uses models. By *engineering*, we mean any kind of engineering: software, firmware, hardware, systems, safety, security, etc. *Models* are conceptual abstractions of physical or digital phenomena.

Marketing information about products and services often talks about MB**S**E, which is ambiguous. MBSE can mean Model-Based *System* Engineering or Model-Based *Software* Engineering.

MBE *augments* other forms of engineering, like those mentioned above, and is *mandatory* for verification of high-assurance systems, and *complements* digital engineering with digital twins.

3.1 Basic Facts of Model-Based Engineering

MBE is not a panacea, a silver bullet, or a joke. Its use on a project says very little, other than the project team uses moderately advanced development tools.

MBE often means, at least, that (1) some abstractions are written down that describe a system under development, (2) repetitive, boilerplate code that programmers are prone to get wrong is automatically generated, and (3) the environment in which a system operates and evolves is taken into account.

3.2 Engineering in the 2020s

Engineering today, especially for embedded systems, looks a lot like it did in the 90s. Programmers like to program, after all, and documentation is often out-of-date.

Unfortunately, the abstractions taught in CS, ME, and CE programs—state machines, grammars, regular languages, and more—are rarely explicitly used

by programmers to create systems.

The C programming language and antique ISAs (e.g., x86 and ARM) are in widespread use, and hardware design languages (HDLs) standardized twenty years ago (e.g., SystemVerilog) are still not widely adopted.

Moreover, in our experience, hardware engineers are loath to learn new HDLs (Chisel, BSV, SystemC, many others) because hardware design firms are extremely risk adverse. Most firms would rather keep using something that has worked for decades than take a shot at a new technology or design flow, even if it might save money, decrease effort, or increase quality. To put a point on it: hardware engineers still use hardware print statements, debuggers, and laborious testing to check correctness.

Rigorous specifications—even boring old assertions—are unheard of, especially anything formal, such as the Java Modeling Language or SystemVerilog Assertions (SVA). In fact, it is fair to say that the vast majority of projects never use, or perhaps have even heard of, the MBE-centric Design-by-Contract methodology introduced in the early 1990s.

3.3 Novel Improvements

Over the past two decades there have been a number of technology improvements that has led to the ubiquity and impact of MBE. In the early days of MBE (back in the 1990s), MBE was often viewed as being too demanding for mainstream computing, both computationally (it impacted performance too much) and resource-wise (it used too much memory or storage).

Fast forward twenty-odd years, and hardware is faster and cheaper than ever. Who would have thought that pre-teens would carry around a multi-core, multi-gigahertz device (a supercomputer!) in their pockets?

New programming languages that are great for safe, embedded software engineering are breaking through, such as the Rust programming language. It is no longer necessary to implement embedded systems in assembly language or C in order to achieve high performance.

A new open, unencumbered Instruction Set Architecture (ISA) called RISC-V has catalyzed an explosion of novel hardware design for the masses. There are dozens of RISC-V CPUs and SoCs on the market, and the software ecosystem behind RISC-V is as large, rich, and stable as that of any other mainstream ISA.

RISC-V, in tandem with the rise in small, community-backed projects like those found on Kickstarter and via CrowdSupply, has created a new generation of open hardware developers. *Open hardware*, the brethren of the open software movement, has even become respected in many industry and government circles, including in U.S. Government agencies that are particularly concerned about reliability (e.g., Sandia National Labs) and security (e.g., the NSA).

Complementing the rise in open hardware, *reprogrammable* hardware (e.g., FPGAs) is widely available, well-understood, and adopted. DoD platform architectures today regularly witness the use of FPGAs to replace what was once

realized by expensive, underutilized general purpose processors. These new hardware-based architectures are higher performance and have much better power-performance ratios than their software predecessors.

Finally, Domain Specific Languages (DSLs) and Domain Specific Architectures (DSAs) have proliferated and increased productivity and quality.

3.4 Typical Set of MBE Technologies

The core set of MBE technologies that are widely used includes modeling/specification languages, modeling environments, programming languages, reasoning tools, and operating systems. Here we enumerate the most prevalent of these that are in use today.

3.4.1 Modeling Languages

Classical "modeling languages" are used to describe software systems and their environment. The dominant three modeling languages of this ilk are:

- **UML: Unified Modeling Language.** UML is mainly used to model software systems, both at a high level (e.g., domain models, requirements, viewpoints, etc.) and at a design level (e.g., classes and modules and their interfaces and APIs, and dynamics of a software system). Some UML tools permit the generation of software from specific UML artifacts (e.g., class diagrams), or the extraction of such artifacts from source code. UML was introduced in the mid-90s and is still in use today.

- **AADL: Architecture Analysis & Design Language.** AADL is mainly used to model embedded, sometimes real-time, systems, including their software, firmware, and hardware components. AADL tools often include powerful reasoning features and some include code generation. AADL was introduced in the early 00s and is in use in mission-critical sectors.

- **SysML: System Modeling Language.** SysML is used to model complex systems at a high-level, including their environments, software, firmware, hardware, and physical manifestations. SysML version 1, introduced in the early 00s, is a *UML profile*, and thus rests on UML as a foundation (for good and ill). SysML version 2 is under development at this time and decouples itself from UML. It should see standardization in 2023 and adoption soon thereafter.

3.4.2 Modeling Environments

Modeling environments are tools that permit the creation of, editing of, generation of, and reasoning about, models. Some modeling environments support only a single modeling language. Others support multiple languages, usually through some kind of extension mechanism. Commercial environments often

costs hundreds or thousands of dollars per-seat and their cost is usually not public, but is instead negotiated on a case-by-case basis by their vendors.

The dominant environments in use today are as follows.

- **IBM Rational Software**
- **Dassault Systèmes Cameo Systems Modeler**
- **Eclipse Software Foundation's Eclipse**
- NI LabVIEW
- Vitech's GENESYS
- MathWorks System Composer, **MATLAB**, and **Simulink**
- Sparx Systems Enterprise Architecture
- **OpenMBEE**
- Ansys ModelCenter and **SCADE**
- BigLever onePLE

Those environments and tools in **bold** are de facto in various government agencies and large engineering firms with which we are familar.

3.4.3 Programming Languages

There are a host of programming languages widely used in the MBE domain.

The two main sub-classes of languages are:

- various subsets of C (MISRA, verifiable), or
- safety-centric languages such as Rust, Safety-critical Java, and SPARK.

3.4.4 Reasoning Tools

There are hundreds of MBE reasoning tools available today.

The main companies that develop and support such tools are: AbsInt, Galois, Microsoft, GrammaTech, Perforce, Parasoft, MathWorks, SonarSource, Ansys, Cadence, Synopsys, and Siemens.

3.4.5 Specification Languages

There are a few dozen main specification languages in use today. These are, in the main, domain specific languages which permit the detailed specification of the behavior or structure of a system, and many have an underlying semantics (or two).

The main languages in use today includes: MATLAB, Simulink, SCADE, Statecharts, Java Modeling Language, CodeContracts, Cryptol, and SystemVerilog Assertions.

3.4.6 Operating Systems and Related Technologies

There are just a few dominant operating systems (OSs) used in MBE. While mainstream OSs like various BSD flavors, Linux, macOS, and Windows are used as development platforms, and sometimes deployment platforms, for MBE, more often one sees MBE used for embedded systems engineering where such heavyweight OSs are ill-fitted.

The main go-to realtime/embedded OSs come from proprietary vendors such as Alteronic, Green Hills Software, Lynx, Segger, QNX, Wind River, or are open source and come from foundations—such as the Apache Foundation—or communities with corporate support, such as FreeRTOS/SafeRTOS—which is now supported by AWS—or MuttX, RTEMS, Zephyr.

Operating systems are complemented by various hypervisors, microkernels, and similar underlying technologies, some of which are widely used even without operating systems. The best examples of these technologies in deployment are seL4, MirageOS, Xen, QEMU, and KVM.

3.5 MBE Technologies that Really Work

Just because a technology is mentioned earlier does not mean that it is fit for use in developing a high-assurance system, such as the control and protection subsystems of a nuclear power plant. In fact, some MBE technologies (e.g., the use of UML without a model-code connection or the use of OCL contracts) are viewed as being more trouble than they are worth in some settings, particularly if a team is not fully bought into MBE.

The kinds of MBE technologies that we see in the field as really working well and are effective for saving time and effort and increasing quality are as follows.

- data modeling and database generation, evolution, and management (Ruby on Rails and other tools),
- user interface design and behavior (e.g., Apple's Xcode, numerous tools/frameworks for Web 2.0)
- lexer and parser generators for language design (e.g., ANTLR, yacc/bison, flex/lex, XText, MPS)
- algorithm and protocol specification and reasoning systems (e.g., Cryptol, F, Ivy, ProVerif, CryptoVerif, FDR)
- Design-by-Contract and assurance of software (SPARK, JML, CodeContracts, Dafny, Kotlin), and
- Design-by-Contract and assurance of hardware (SVA, BlueCheck, OneSpin, Jasper, Formality, etc.)

3.6 MBE for High-Assurance Engineering

Very few MBE technologies with a focus on mission-/safety-critical systems have been commercialized. Those that have have fairly small customer bases of dozens to hundreds of individuals/firms, rather than two or three orders of magnitude more, as is the case with mainstream development tools.

Numerous MBE technologies exist that are applicable to high-assurance engineering, but have no corporation solely supporting their use, and are mostly supported by ICs and SMEs. Examples of tools and technologies that are suited for high-assurance engineers are as follows. Those that are blessed and widely used in the DoD are highlighted in **bold**.

- design environments from **Dassault**, CMU SEI (**OSATE2** and Oquarina), Vitech, Sparx, **IBM**, **Ansys**, **Siemens**, Visual Paradigm, PTC, etc.
- reasoning tools from **AbsInt**, **Galois**, **GrammaTech**, **Ansys**, Runtime Verification, etc.
- logical frameworks such as **Coq**, **Isabelle**, **Lean**, and **PVS**,
- classical formal methods such as B, Event-B, VDM, and Z,
- modeling intermediate representations such as **Boogie**, **LLVM**, **ZIMPL**, **SAT**, and **SMT-LIB**,
- constraint, SAT, and SMT solvers such as **CVC**, **MiniSAT**, **Yices**, **Z3**, and many others,
- open source development and modeling environments (**OpenMBEE**, **Eclipse**, **VS Code**, etc.), model checking tools (**Alloy**, **CBMC**, **SPIN**, UPPAAL, FDR4, **SPIN**, **TLA+**), hardware verification tools (**QED**, **Yosys**, etc.), and
- numerous MBE technologies have been funded by the DoD, IC, and NASA such as **Cryptol**, **SAW**, **Crux**, **AADL**, **CASE**, **ARCOS**, **CoPilot**, **AdvoCate**, etc.

The state-of-the-art in MBE in 2022 outside of the mainstream commercial realm is mainly found in the list above. These tools are mainly focused on DSLs and DSAs (e.g., the model checking tools mentioned above, Cryptol, and SAW), they include model-lifting and reasoning features (e.g., SAW and Yosys), code generation and reasoning features (e.g., Cryptol and SAW), model-based runtime verification/testing, BISLs and model-based software engineering (the aforementioned JML, SPARK, etc.), or HLS/UVM/SVA for model-based hardware engineering.

3.7 Gap Analysis

Today's MBE tools have seen thirty years of evolution, are powerful, very useful in commercial product development, often save developer and development time and money, and continue to evolve to work with new platforms, languages, and tools. Even so, there remain many practical and research gaps between today's tools and the vision of wholly integrated, high-assurance model-based enginering. This section summarizes many of those gaps, particularly with regards to the demands of high-assurance, safety-critical systems engineering.

3.7.1 Market Gaps

As discussed in section 3.6, there are insufficient customers in high-assurance engineering to support a rich ecosystem of tool providers, platforms, and products.

Part of the problem is that, in the large, software engineers expect tools to be free, and historically virtually all commercial model-centric tools have been prohibitively expensive. Thus, most software engineers have little-to-no experience with using high-end commercial tools, and thus do not appreciate their utility.

Additionally, there is a pipeline problem in the discipline. While MBE has been around for thirty-odd years, few students graduate with knowledge of MBE because most universities have very few to no courses in the topic.

Corporate adoption of MBE also holds back the field. As with many engineering disciplines whose focus is largely on making money, the good is the enemy of the best. While MBE is used successfully by many companies as a market advantage, it does not see widespread use because the market awards first-to-market products, many firms get away with creating throw-away products poorly and quickly, and thus eschew not only models, but any other artifacts that help with long-term maintainability or correctness, such as documentation and rigorous testing.

Another challenge in the market is that of usability. Easy-to-use products are much more widely adopted than harder-to-use-but-more-impactful products, and unfortunately, none of the existing high-end MBE products seem to have seen any user-centric design.

Historically we have seen some IEEE/ISO standards and specific tools adopted due to compliance and certification mandates (e.g., SCADE or AbsInt tools for development that is meant to be assured under FAA DO-178). But compliance and culpability are not (yet) forcing functions in the world of MBE because no large agency or program has mandated their use. This may change soon, with the evolution of the U.S. Government's Digital Engineering strategy and modernization goals across the DoD, but for the moment, major DIB firms are largely holding pat on development tools, processes, and methodologies of two decades ago.

Finally, mission-/safety-critical industries are risk adverse. MBE, while decades old, represents a fundamental shift in the means by which systems

are created. Adopting MBE means developers learning at least one or two new languages, development platforms, testing frameworks and methodologies, and verification tools. Shifting a workforce to a new language is hard enough; shifting to a whole new way of thinking about design, implementation, validation, and verification is another thing entirely. Consequently, we should expect to only see new entrants and greenfield products using MBE, rather than large, old, traditional firms.

3.7.2 Research Gaps

There are just over a dozen widely-recognized research challenges that have stifled more widespread adoptions of MBE over the past few decades. Many of these challenges are the focus of specific academic and industrial research programs, particularly at Galois, and historically in the academic career of one of the authors (Dr. Kiniry).

Each research challenge is only tersely summarized here, as explaining each of these in any degree of detail represents a section of material.

1. usable semantics that refine between levels
2. usable and useful traceability with semantics
3. reversibility of traceability
4. ambiguities in interpreting English into models
5. connecting semi-formal and formal models
6. consistency between text and graphical models
7. specifying and reasoning about systems of systems
8. supporting MBE in an agile methodology
9. augmenting IDEs with MBE for continuous learning
10. integrating rigor without loosing usability and scale
11. supporting product lines throughout the lifecycle
12. predictable and deterministic environments and products
13. consistency of all models across the whole lifecycle
14. trade space analysis and measurable, useful metrics

3.7.3 Practice Gaps

Finally, there are a number of nuts-and-bots practical gaps in widespread adoption of MBE, especially for the mainstream. These gaps are not a hindrance to the use of MBE for mission-/safety-critical systems, as today's developers in those domains are rewarded for thinking hard and being careful, unlike your average web programmer.

Writing models is rarely as immediately rewarding as programming, since most are not executable. Models are often denotational as well, and thus are hard to think about and write for programmers who are used to thinking and "doing" operationally.

In fact, using modeling tools often feels more like writing and reviewing documentation than programming, and, frankly, programmers like to program. Some of the most common techniques used by programmers to understand and experiment with their systems, such as the infamous inline "print" statements, are rarely seen or used in modeling languages. And some modeling languages are only graphical, whereas most programmers prefer textual languages.

The biggest practical gap to widespread adoption in the broader set of mission-critical systems across the U.S. Government is historic. Trillions of lines of legacy code are not described with any existing models, and extracting models from code is much harder than generating code from models. Frequently, agencies and branches do not even have copies of the source code for the binaries that are in platforms. While lifting models from binaries is possible, it is the focus of several current DARPA projects, and thus is largely not ready for transition to mission-/safety-critical systems.

Chapter 4

Reviewing Model-based Systems

This chapter summarizes best practices in reviewing, validating, auditing, and certifying systems developed with model-based processes, methodologies, tools, and technologies.

4.1 Reviewing

Systems that have been developed using MBSE tools are typically delivered in one of two ways: directly through project access in a collaborative development environment (e.g., a CDE like GitHub, GitLab, etc.) or an archived snapshot/release of the system.

The standard development artifacts that come with such a delivery are:

- digital documentation (e.g., PDF, HTML, ASCII text files, etc.),
- models (stored in either open or proprietary formats),
- model views (interactive or static figures generated from models),
- source code (written in software programming languages or hardware definition languages),
- physical hardware designs (such as a PCB layout or a GDS-II file describing an ASIC),
- binary representations (object files, executables, shared and static libraries, bitstreams, and other document file formats), and
- containers used to, in turn, store stable snapshots of development, validation, and verification environments (such as a Docker image).

4.2 Tool Dependencies

Many of these artifacts are easily examined with off-the-shelf tools that come with standard operating systems, such as text editors, web browsers, and PDF viewers. But, given the nature of MBSE, one often finds dozens, or even hundreds, of artifacts stored in non-traditional file types. In order to read, evaluate, understand, and validate those files, one has to have the appropriate tools available.

Commercial MBSE tools commonly define their own proprietary file formats, and these formats change over time in backwards incompatible ways. Consequently, in order to vet a model artifact created with a proprietary tool, one often has to have access to exactly the right version/release of the tool that was used to create the artifact in question.

Recommendation 1 (Tool Metadata)**.** *For each artifact kind and file type in a system under review, ensure that the tool(s) necessary to validate that artifact are precisely documented (name, vendor, version, operating system platform).*

Recommendation 2 (Tool Availability)**.** *For each artifact kind and file type in a system under review, ensure that the tool(s) necessary to validate that artifact is made available to the reviewer(s).*

The best practice in fulfilling recommendation 2 is to deliver with the system an evaluation image snapshot that includes all necessary tools "out of the box".

Recommendation 3 (Evaluation Platform)**.** *For each system under review, the system development team should provide a snapshot of a stable, coherent, documented evaluation platform to reviewers, such as a Virtual Machine image (e.g., VDI, VHD, VMDK file types), a Docker image, or a Nix configuration and cache.*

4.3 Trust, but Verify

When reviewing a system, one typically examines artifacts in a top-down fashion, moving from most abstract to most concrete. Whether or not one pursues a breadth-first or depth-first approach in moving through model abstraction layers is up to the reviewer's personal taste. Different people think in different ways when it comes to taking in and understanding a system that is larger than one can easily grok in an hour or less.

Moderate to large scale systems often include far more artifacts than can be reasonable examined in detail by a reviewer or review team in an acceptable amount of time. In such a circumstance, reviewers commonly pursue a "Trust, but Verify" approach to examining artifacts.

There are three key facets to such a review methodology, and they are summarized in the following recommendations.

Recommendation 4 (End-to-End)**.** *Review at least one most abstract to most concrete chain.*

Recommendation 5 (Regularity)**.** *Review at least one instance of each artifact kind in detail, and check for regularity across adjacent model elements of the same kind.*

Recommendation 6 (Workflow)**.** *The recommended workflow for reviewing model-based assets moves through the four key facets of any system specification: structure, then data, then behavior, and then properties.*

The reason that the ordering in recommendation 6 is recommended is because, typically, dependencies flow backwards through this ordering, and thus, e.g., in order to understand behavior, one must understand data and structure.

Recommendation 7 (Limit Risk)**.** *Review remaining artifacts (of all kinds, model, code, documentation, etc.) using a* risk-limiting audit *approach.*

4.4 Key Relations

Examining only artifacts such a models, source code, and documents is insufficient for validating or certifying a system created with MBSE. The *relationships between artifacts* are often as or more important than the artifacts themselves.

There are several key kinds of relations between artifacts to look for when examining a system. The following list, while extensive, is by no means complete.

Traceability. Traceability relations are those that link artifacts, often in a *bi-directional* fashion. Traceable relations should be labeled to indicate the purpose of the reference.

By bi-directional, we mean that many useful traceable relations define a trace relation $\mathcal{T}$ between two elements A and B as $A\mathcal{T}B$ and, by virtual of its bi-directionality, one can follow the trace *from* A to B and, independently, from B to A.

Traceable relations commonly found in documentation include glossaries, indices, table of contents, lists of figures, lists of table, etc. Best practices in traceable relations enable both *active tracing* via hyperlinks, hovers, or other similar simple action triggered by a mouse or keyboard event and *passive tracing* via a traditional unique directional reference, such as a page number or file name with a relative or absolute path.

Traceable relations commonly found in model-based systems engineering artifacts include requirements and feature matrices, model-model/model-view/model-code relations as described below, abstract/concrete refinements, definitional lookups, and evidence derivation.

Recommendation 8 (Traceability Completeness)**.** *At every layer of model refinement, every assumption, goal, and requirement that frames a model-based specification at the abstract end of the refinement must be traceable to a corresponding assumption/rely/precondition, goal, or property in at least one refined artifact.*

Without traceability completeness, model elements are left "hanging." For assumptions this is not fundamentally problematic: nothing needed the assumption in order to build and assure the system. But for goal and requirements, and artifacts derived thereof, such as scenarios, events, properties and the like, it is much more problematic because, if such a traceablilty relation does not exist, then the goal or requirement is unfulfilled in the assurance case.

Recommendation 9 (Bi-directional Traceability)**.** *Every model, code, or assurance artifact A that is in a traceability relation to another artifact B should (i) permit a trace from A to B and from B to A, and (ii) the nature of both traces must be clear.*

Uni-directional traceability, while somewhat useful—as it help justifies other traceability properties like completeness and reachability—is not satisfactory from a reviewer point of view. One should be able to trace each artifact *to* its more abstract dependency (be it concretized as a requirement, goal, invariant, etc.) and *back* from that artifact. Likewise for any relations *down* from the artifact to more refined artifacts.

Recommendation 10 (Traceability Reachability)**.** *Every model, code, or assurance artifact must be reachable by a trace relation to at least one, and preferably only one, other artifact.*

Every mode, code, or assurance artifact that is included in a system built with MBSE technologies should be there for a reason. A traceability relationship to each artifact is, in part, the means by which to justify its inclusion.

Recommendation 11 (Traceability Implicitness)**.** *Implicit traceability is superior to explicit traceability.*

Explicit traceability is realized by the writing or generation of model annotations which document trace relations. These annotations are often in the form of code or model comments, model notes, extra spreadsheet or CSV columns, source code formal annotations, etc. In our experience, such annotations are fragile, clutter models and programs, and are very difficult to maintain in the face of evolving artifacts on both ends of the traceability relationp.

Implicit traceability is realized by the use of types, naming convensions, and methodological rules that are machine checkable and human understandable. In model-based systems that use implicit traceability, no extra comments, notes, etc. are necessary, as one can always trace up and down between abstraction layers in a system design and implementation.

Tools, such as IDEs with rich user experiences, can assist with such tracing, such as the ability to lookup a type declaration and its semi-formal definition by touching a variable in a model or code and seeing a pop-up help message that can be activated to jump to the definition in the current or new window. Refinement or traceability checkers can analyze a system to determine if a system's implicit traceability is complete, bi-directional, and if all model elements are reachable.

Representational. Representational relations connect model abstractions to physical, digital, or computational manifestations. These kinds of relations are often based upon model abstractions that are meant to represent real-world constructs, whether they are physical (such as a CAD model of a PCB design), digital (such as a GDS-II model of an ASIC mask, a Entity-Relation model used to describe a database schema, or a wireframe model used to describe a user interface), or computational (such as an algorithm described in pseudocode or a protocol described with a domain specific language).

In each case, the model is meant to be an abstraction, and thus *every* property the model has should be a property witnessed by the thing it describes. In such a relation holds, we call the model *sound* with regards to its concretization. Moreover, a *complete* representational model is one which captures *all* properties of interest, given the context and goal of the model.

Recommendation 12 (Representation Realizability)**.** *Every model must be realizable within an implementation; that is, once the model is refined to an implementation, an implementation must be able to be exist which fulfills all of the properties of the model. A model that is unrealizable is a model that is unimplementable.*

Sometimes models are created without a thought to the implications of the model's properties. For example, a model might demand arbitrary amounts of compute or storage resources in order to be implemented. Or a model might demand that two properties hold at the same time, and those two properties contradict each other. This is very commonly seen when translating informal requirements into formal requirements, as English is notoriously imprecise and few write requirements with an eye toward logical consistency. All of these circumstances lead to unrealizable systems. Tools exist to help debug some of these kinds problems by, e.g., finding minimal models that resolve an inconsistency.

Recommendation 13 (Representation Soundness)**.** *Models must be sound; unsound models are neither useful nor realizable. Models and model-code relations should be checked for soundness by attempting to rigorously validate or formally verify "bottom" implementations.*

Demonstrating that a model is sound is often overlooked, but is necessary. During model evolution is is quite common to accidentally make a model unsound, and without some means by which to "smoketest" soundness, this can lead to a large amount of lost effort, late surprises in verification, and project risk.

The easiest means by which to check the soundness of a model is to intentionally attempt to demonstrate (either through runtime assertion checking or formal proof) a false proposition; i.e., try to prove that the lemma *false* holds. One can accomplish this in programming languages which provide an assertion construct by simply asserting *false*, thereby saying "the program that crashes, terminates successfully." Galois implements a "bottom" version of all models that does just this, and vets the soundness of models by attempting to rigorously validate or formally verify the whole of *bottom* with multiple tools.

Recommendation 14 (Representation Completeness)**.** *Every model should be* relatively complete*, where relative completeness is defined with regards to (i) the expressivity of the modeling language, (ii) the utility and capability of tools which can reason about the model, and (iii) the property coverage necessary based upon the refinement relationships in which the model participates.*

While relative completeness has a formal meaning, and while it does connect with the intention of this recommendation, it is rarely the case that our notion used here of the same name is defined as such.

In essence, each modeling language, and the tools that support operating on or with models expressed in the language (checking, reasoning about, compiling, synthesizing, etc.), is good at modeling a certain kind of *thing*, and that thing's *properties*. Those are the kinds of things that one should endeavor to cover in any model expressed in that language. If there are simple concepts and properties which should be included in a model, but are not, that is a warning sign with regards to relevative completeness.

Likewise, there are often certain shaped properties which, while possible to model in a given modeling language, are *unreasonable* with current tooling. By *unreasonable* here we mean that state of the art tools, while able to provide a semantic interpretation for an unreasonable property, cannot effectively determine if the property holds or not.

An abusive example of such a property is the embedding of an unsolved or extremely difficult theorem (say, Fermat's last theorem) into a specification. Another formal example, not uncomonly seen, is an attempt to formally verify a correctness property of an infinite stream without a reasoning framework that supports coinduction. A third common example is the statement of temporal logic expressions with prolific use of the *leadsto* operater, which essentially says "this thing will happen someday, but it could take a **very** long time."

Finally, if a (more abstract) model A is refined to a (more concrete) model B, and property P_A holds on A, then the refinement of P_A should hold on B. If it does not, then the model is relatively incomplete with regards to its abstraction's properties.

Descriptive. A descriptive relation is a relation that also connects an abstraction to a descriptive target, but no property correspondence is meant to hold between the model and the object it describes.

Common examples of descriptive relationships include document substructure summaries (e.g., a paper's abstract, a chapter's summary in an introduction, or a book's jacket description), source code substructure summarizes (a module's general purpose or a function's general meaning, as described in structured source code comments), and alternative text describing images for the sight impaired.

Recommendation 15 (Descriptive Accuracy)**.** *Descriptions should be accurate, as judged by a reader who is familiar with its target.*

Inaccurate descriptions are worse than no descriptions at all. The most common example of inaccurate descriptions is natural language comments on source code and system descriptions (architectural, security, operational, or otherwise) of systems that have witnessed a lot of evolution. In both cases, developers are wont to rewrite code, and rarely attend to updating the corresponding documentation, even though writing the documentation takes a fraction of the time taken to write and assure the code.

Recommendation 16 (Descriptive Completeness)**.** *Descriptions should be complete with regards to the relative completeness of the artifacts to which they refine.*

A description, while informal, has a scope driven by any formal properties associated with the artifact that is being described. Thus, the description should cover those properties/features of the artifact that are salient to the model. For example, if a model-based protocol specification attends to reasoning about protocol properties like robustness or fairness, then its description should discuss robustness and fairness. Likewise, if a concurrency model is created in order to reason about absence of deadlock and livelock, then its description should discuss such.

Structural. Structural relations are *part-whole* relations of various kinds. They are prolific in any artifact that has a hierarchical decomposition, such as this report, a system's source code, or a complex formal model.

Part-whole relations come in several flavors, including the dual expanded and referential substructures, containment, ownership, physical position and orientation, token-based access control, and more.

Recommendation 17 (Structural Property Preservation)**.** *All traceable refinements of models that have structure should preserve all structural properties.*

Note that recommendation 17 does not prevent the many ways tools transform models or code in ways that typically munge representations. A few examples of transformations that are, surprisingly, often struture-preserving: source code optimization during compilation via inlining; model flattening when moving from an very expressive (e.g., higher-order) model to a less expressive one (e.g., first order), transforming a operational artifacts, such as LLVM bitcode, into a relational artifact, such as a first order symbolic evaluation; and flattening a nested reference model by expansion of nested elements.

Operational. Operational relations are relations that describe possible unconstrained behaviors. One side of the relation often characterizes a set of possible abstract operations that can take place in a model, and the other side of the relation describes the concrete operations (such as function calls, message send/receives, UI events, etc.) that correspond to those abstract operations.

Classically, these kinds of relations are used to describe a variety of model phenomena, such as non-deterministic behavior, concurrency, timing uncertainty, human-in-the-loop actions, adversarial responses, etc.

Recommendation 18 (Operation Realizability)**.** *Operations must be realizable in a system.*

This recommendation is linked with the other, more abstract, recommendation that focus on realizability (recommendation 12). Operations, as the concretization through refinement in models or code of computation, must also be realizable, otherwise the system cannot be built.

Recommendation 19 (Operational Elegance)**.** *Operation design and implementation should be* elegant*. Elegance means: (1) there should be only a single operation that fulfills any system feature, (2) each operation should only implement a single feature, (3) simpler operational specifications are superior to more complex ones, (4) operations should either be pure queries (that change no system state) or simple commands that return no information beside success or failure, and (5) an operation that is both a query and a command (it changes state and then returns information) should be specified and implements as the composition of those two existing operations.*

Many of these recommendations come from rules derived from verifiable engineering practices, as initially espoused by Meyer, and inspired by best practices from formal methods like B, VDM, and Z. Tools are available that will analyze elegance factors like those described above, both at the model and code level.

A high-quality MBSE methodology will include the use of such tools to automatically check for model and implementation elegance, as doing so ensure that this property is preserved during development and post-release system evolution, and it also helps ensure that the system's assurance case is no more complex than it needs to be.

Recommendation 20 (Operation-Action Coupling)**.** *Every system operation should refine to a single system action.*

This recommendation is the direct implication of the first two clauses of the definition of *elegence* in recommendation 19 and recommendation 17. Because it is an important system property, we highlight it in its own recommendation.

Data. Data relations describe models of data and their relationship with data in implementations. Typically, data in models is described abstractly, using generic mathematical objects like sets, sequences, vectors, streams, etc. Data in applications is, on the other hand, concretized in programming language data types and/or serialized data, such as records in a database or network packets in a transmission.

In order to relate abstract data and concrete data, a data relation must explain what invariants hold on data types, and especially *between* data types, and how those invariants are maintained by the implementation.

Recommendation 21 (Absent Invariants)**.** *It is difficult to tell the difference between a component that has* no invariants *and one that has* absent invariants*.* Absent invariants *are invariants that hold for a model, but are not expressed by the development team due to accidental omission, oversight, or neglect.*

No component should have absent invariants, and the best way to document such is to have process or methodological conventions that permit an observer, be it another developer or a reviewer, to understand that the invariant for a component with regards to a particular property really is just *true*.

Recommendation 22 (Data Simplicity)**.** *The* core data model *of a system should be modeled in the simplest possible way, and model or implementations complexities due to optimization, representation, platform, or infrastructure should be modeled as data refinements of the core data model.*

Recommendation 23 (Data Portability)**.** *Data portability should be accounted for in a model using a platform product line specification, thereby factoring out platform-specific issues from core system specification matters.*

Recommendation 24 (Data Transparency)**.** *Data models and their representations should contain no undocumented values, interpretations, structures, or relations. If the semantics of data fulfill this condition, it is* transparent *to any developer or reviewer.*

Recommendation 25 (Data Explainability)**.** *The semantics of data should be* explainable *within a model refinement; every data invariant should be justified by a traceable refinement* up *to a more abstract model property, or* down *to a more concrete model or system property.*

Action. Finally, action relations describe the decomposition of complex actions into simple events, and how those actions and events are realized in a system implementation. An event is a state change in a system that, from the point of view of the observer, is atomic. An action is set of a sequences of events, possibly just a linear sequence (first *this* happens, then *that* happens), but often, instead, a branching tree of possible events. Actions are also known as *scenarios* and *use cases* in some methodologies.

An action relation maps an abstract action and its events to a concrete system. A *sound* action mapping ensures that any action that can be witnessed in the model can, in turn, be witnessed in the system, and, preferably, those two actions are in a refinement relation. A *complete* action mapping ensures that every interesting action (as defined by the domain of specification in the abstraction) is included in the model.

Actions are often used to describe both the normal behaviors of a system—how it should act under typical circumstances—and the abnormal behaviors of the system—how it should act in the presence of failures. Additionally, action specifications can describe non-terminating behavior, such a system that is meant to run continuously without fail.

Recommendation 26 (Positive Action Completeness)**.** *Actions that define the behavior of a system under normal circumstances (no errors, exceptions, or faults) are* positive actions*. The model-based specification of a system should be (representationally)* complete *(see recommendation 14) with regards to positive actions.*

Recommendation 27 (Negative Action Completeness)**.** *Actions that define the behavior of a system under abnormal circumstances (errors, exceptions, or faults) are* negative actions*. The model-based specification of a system should be (representationally)* complete *(see recommendation 14) with regards to negative actions.*

Note that (i) no positive action can be equivalent to a negative action, and (ii) the union of all positive and negative actions is the sum total of all actions defined on a system.

Recommendation 28 (Positive-Negative Consistency)**.** *The composition of positive and negative actions must be consistent, and thus realizable.*

Recommendation 29 (Action-Implementation Consistency)**.** *For every action A, A's action-event structural relation (that is, each action is decomposed into a set of events) should be maintained by the model refinement or implementation of A.*

This recommendation is a corollary of recommendation 13 and recommendation 17.

Recommendation 30 (Action-Event Completeness)**.** *Every event must be used in at least one action, and realized by a single model refinement or implementation feature.*

This recommendation is a corollary of recommendation 8 and recommendation 14.

Relations in Systems Modeling Languages. Each modeling language includes some subset of these modeling relations. Rich systems modeling languages, such as SysML, cover virtually all of these relation types. Other, more restrictive, modeling languages, especially Domain Specific Languages (DSLs), only cover a small number of relation types. For example, entity-relation models and basic SQL models are solely about data, and thus only focus on data relations.

A well-designed modeling language will ensure that its relations fulfill many/all of the relation recommendations in this section, and, furthermore, that tooling should support the creation, update, checking, and enforcement of such relations and their desirable properties.

Recommendation 31 (Relation Typing)**.** *For each relation identified or discovered in a system created with MBSE, if the relation is not explicitly typed according to the above classification scheme, deduce its type and check if it has the corresponding recommended properties described above. If it does not, ask the developers to justify its partial properties.*

Asking for justifications often forces developers to use the full existing features of their modeling languages and tools and to update their process or methodology to guarantee certification demands, such as those recommended in this section.

4.5 Validating Models

In order to validate a model, one must work to understand the model's *purpose*, *utility*, *context*, *relationships*, *history*, *evolvability*, and *maintainability*.

Each of these model facets has at least a subjective, and sometimes and objective, quality measure. Determining the means by which such metrics are measured, one can "grade" each dimension as a part of a model validation process. We give some examples below for the metrics used at Galois for this kind of evaluation.

Recommendation 32 (Model Purpose)**.** *Every model should have a* purpose*—a reason for it to exist.*

Typically a model's purpose is to describe something that is not described elsewhere, or is indescribable in the other models used to specify a system. Another good purpose is to describe a system property from another perspective, either to transform a property in a such a way that a second reasoning tool can revalidate/reverify the property (see recommendation 46 below), or to explain the point of a model artifact or property to a developer or reviewer (cf. recommendation 15).

Recommendation 33 (Model Utility)**.** *Every model must have a* utility*: a means by which to use the model toward providing assurance for a system.*

Utility is typically either semi-formal—the model is used to explain a system property to a developer or a reviewer—or is formal, and is used to rigorously validate or formally verify that system properties hold (cf. recommendation 17 and the recommendations of section 4.7).

Recommendation 34 (Model Context)**.** *Every model must have an explicit* context*: a frame of reference in which the model's assumptions/rely are explicit.*

A model without context is missing information critical to developers, reviewers, and reasoning tools. While on some rare occasions a model needs no context—it makes no assumptions at all outside of those definitionally included in the model—most models have an enormous number of subtle assumptions in order for the model to be valid. An example of a simple unspoken model assumption that is frequently seen in embedded systems code is that there is an unspoken assumption that no pointer used in any function invocation is `NULL`.

Assumptions that are made between refinement levels or between system abstraction layers are known as *relies*, as one can often use a form of specification and reasoning known as *rely/guarantee reasoning* in order to sensibly reason about the composition of layers in an architecture.

Recommendation 35 (Model Relationships)**.** *Models are often connected with one another through relations, such as those described in section 4.4. A reviewer must understand and validate every model relationship, and every model relationship should have a purpose (cf. recommendation 32).*

Recall that model artifacts include not just components and their features (i.e., attributes, functions, procedures, events, signals, etc.), properties (goals, requirements, invariants, etc.), and aggregations (i.e., subsystems), but also all functional and relational connections **between** model elements. This includes even, e.g., relations between relations, or functions on modules.

Recommendation 36 (Model History)**.** *A reviewer should be able to observe the history of every model artifact, as that history often tells a story about the design and modeling decisions made by developers during a model's evolution, validation, verification, and refinement.*

Model history is often captured in one of two ways: (1) via a built-in design justification history mechanism, as provided in advanced MBSE IDEs, or (2) via a revision control system changelog when used with a rigorous development process.

Recommendation 37 (Model Evolvability)**.** *All models should be* evolvable *in a relation-preserving fashion.*

If an MBSE developer is fearful of changing any model or implementation artifact because the change will cause a large, particularly manual, change in other artifacts, then the model is not evolvable. It is very common for complex system specified with SysML version 1 to have this property, and thus models are constructed as read-only, "*golden*" *system architecture specifications* or *platform specifications* and then rarely changed thereafter. Such is indicative of a seriously broken MBSE methodology.

Recommendation 38 (Model Maintainability)**.** *All models should be* maintainable *after system deployment.*

Once a snapshot of a system "ships"—whether it is to a client, to a certification authority, or to an integrator—it must then be *maintained*. Maintenance is triggered either through dependencies (e.g., via an upstream change, such as a bug is found in a dependency which forces a model or implementation update), client feedback (e.g., certification rejection with recommendations, or a bug is discovered during acceptance testing), or because of an adversary (e.g., a security vulnerability is exploited which forces a maintenance update to mitigate the flaw).

A model that cannot be maintained in the presence of these common post-deployment scenarios fails this criteria. Such models become archaeological artifacts describing the history of a project or product, and are no longer useful for the product that exists in client's hands *today*. Many antique waterfall-based MBE tools fall into this trap, such as having an accurate UML model of a system in the first month's of a system's development, but the system's implementation today has little or nothing to do with that model just a few months later.

4.6 Validating Model-Implementation Correspondences

The most valuable models are those that have a direct, measurable, checkable relation or correspondence with implementation artifacts.

On many occasions, models are created in order to think abstractly about a system and its properties, and then the model is barely used as a documentation reference when designing and implementing the system. This is an example of poor model-implementation correspondence. If there is little-to-no evidence that the implementation corresponds to the model, then the model has little utility beyond helping developers or architects think about a system.

Note that the situation, while extremely common, is not unacceptable. After all, thinking carefully before diving into the very expensive and time consuming task of building a system often pays dividends: reduced time and effort, lower risk to schedule, etc. But given today's MBSE technologies, there is no reason to not have strong model-implementation relationships, and to keep models and code in sync.

When reviewing a system developed with MBSE from the model-implementation correspondence point of view, we make the following recommendations.

Recommendation 39 (Model-Impl Traceability)**.** *The most concrete models that refine to implementations must be traceable.*

This recommendation is a corollary of recommendation 8, but is particularly critical as the implementation, not the model, is the real system.

Recommendation 40 (Model-Impl Rigor)**.** *The strongest, preferred relationship between a model and an implementation is one that is* rigorous*. A* rigorous model-implementation relation *is one that is codified mechanically (programmatically, in a machine-readable fashion) and can be automatically (re-)checked computationally.*

There are numerous specification languages and reasoning tools which provide rigorous model-implementation relations, such as ACSL/Frama-C, SAWScript/SAW, JML/OpenJML, Eiffel, SPARK, and more. Without a rigorous relationship between models and implementation, model-code relation *drift* must be mitigated using manual, expensive, time-consuming, process-based techniques.

Recommendation 41 (Model-Impl Refinement)**.** *The strongest, preferred model-implementation relation is one that is a* refinement *relation.*

If a model-implementation relation is not a refinement, then earlier recommendations are violated, such as recommendation 17, and there is little clarity with regards to which model properties must, can, should, or must not hold for its implementation. This situation is often hard to understand, review, and is unmaintainable.

Recommendation 42 (Model-Impl Evolvability)**.** *Model and their implementations should* co-evolve*, preferably in an* automated *fashion.*

As a system evolves—whether during development as features and enhancements are added, bugs are fixed, or when requirements change—the system's models and implementation change over time. Every model-code relation is an extra artifact that must evolve as well, and because these relations are often used for traceability and assurance, it is critical that they are kept up-to-date and correct. While such updates can be accomplished manually, doing so if often expensive, time-consuming, and error-prone.

Recommendation 43 (Model-Impl Canonicity)**.** *In every model-implementation relation, either the model or the implementation should be declared canonical, and used as the ground truth for the relation.*

Some powerful tools used in MBSE, such as Ansys SCADE and Galois's Cryptol, SAW, and Crux tools lift models from code or generate code from models. The source artifact of these tools is the canonical artifact fulfilling recommendation 43.

Other tools which do not have this nature typically expect that models and implementations co-evolve, per recommendation 42. A development process and methodology should precisely define and reinforce, preferably with automated tool support, how such evolution takes place.

The two most common rigorous methodologies are *Design by Contract* and *Contracting the Design*. *Design by Contract* holds that the contract (a model that describes a system's architecture, behavior, and more) is canonical. *Contracting the Design* holds that the code is canonical, and updates to the code must be periodically translated into the model.

But by far, the most common approach to attending to model-code conformance is to simply not do it at all. Many projects that use some MBSE technologies only pretend to do so: models have few properties, little-to-no validity or verifiability, and have no relation, semi-formal or rigorous, with the implementation at all. Such projects typically violate most, if not all, of the recommendations made in this chapter.

4.7 Validating Claims

Many claims will be made about a system developed with MBSE tools. A reviewer must validate many or all claims as a part of the review or certification process. The recommendations from section 4.3 especially hold for claim validation.

A *rigorous claim* is a decidable proposition about the system: it is either true or false and there is evidence to bolster either interpretation. An *informal claim* about a system is any statement made about a system for which there is not measurable, decidable evidence. Often such claims are more of the form of *beliefs* of the developers, and evidence for their beliefs are not found in peer-reviewed literature, case studies, or cannot be generated by explainable computational tools.

When evaluating the claims made about a system that was developed using MBSE technologies, we make the following recommendations.

Recommendation 44 (Automated Claim Validation)**.** *Automating claim validation is far superior to, and preferred to, manual claim validation.*

Recommendation 45 (Interactive Claim Validation)**.** *While machine-assisted interactive claim validation can provide as much, or more, rigor, confidence, and assurance of a claim, interactive evidence often requires much more effort, time, cost, and attention than automatic validation.*

A halfway point between full automation and full interactivity is often possible: one can automatically validate claims that were originally made constructed interactively. Doing so permits automation, such as in continuous testing and integration systems and incorporation in developer pre-merge regression checks.

Examples of such a mechanism are in Galois's work in formally verifying Amazon's cryptographic libraries (re-running hand-written SAW proofs of the correctness of C and assembly implementations of high-performance, mission-critical cryptographic functions) or in the re-execution of proof scripts in a logical framework like Coq, Isabelle, or PVS.

Recommendation 46 (Claim Multi-Validation)**.** *Rigorously validating or formally verifying a claim with multiple tools or techniques is what we call* multi-validation*. In general, it is better to witness validation via multiple techniques, and multiple tools which rely upon distinct foundations, reasoning approaches, and different organizations.*

Validating a claim with only a single technique or tool amounts to putting all of your eggs in one basket. By validating each claim using multiple techniques and tools, several kinds of risks are mitigated, such as the discovery of a critical flaw in a reasoning tool or missing environmental assumptions about a platform model in a static formal reasoning technique that might be caught when runtime checking claims on the actual.

Likewise, by using tools from multiple sources/vendors that use different foundations or reasoning techniques increases surety, such as using multiple formal verification tools from competing EDA vendors to assured a piece of critical hardware against a model written using SystemVerilog Assertions (SVA).

Recommendation 47 (Continuous Claim Validation)**.** *Claims should be* continuously validated*, not checked all at once aperiodically with great effort. Preferably claims are (re-)checkable on every developer's development machine, on private or cloud-based compute servers, or in a continuous integration (CI) service (e.g., as a part of a DevSecOps infrastructure).*

Automatically checking claims continuously in a CI service is what we call *Continuous Validation* or *Continuous Verification* (uniformly abbreviated "CV"), depending on the nature of the assurance.

Recommendation 48 (Claim Validation Dependencies)**.** *All dependencies of a claim and its validation should be included in an assurance package, be explicitly or implicitly traceable, and be validated as well, preferably starting from axiomatic foundations and moving up dependencies eventually to top-most theorems and properties.*

Sometimes assurance packages do not include all dependencies explicitly. This is problematic for several reasons, some of which include: (1) the accidental or purposeful misrepresentation of a dependency's nature, claims, and assurance, (2) dependencies that disappear over time (websites go offline, companies go out of business, licensing requirements change), or (3) poor release management of dependencies, and thus it is unclear exactly what version of a dependency is necessary (such is prolific in modern "Web 2.0" development styles that wantonly pull dependencies from the internet).

Recommendation 49 (Safety Claim Validation)**.** *Safety claims should be, if at all possible, validated with formal verification, not (only) with runtime assertion/property checking.*

Safety claims, in the main, cannot be tested for validity, despite this being the most common way that safety properties are validated in system implementations.

Recommendation 50 (Risk Mitigation Claim Validation)**.** *Risk mitigation claims should, if at all possible, be validated using mechanized/computational techniques, and not just with subjective human judgment.*

Recommendation 51 (Hazard Mitigation Claim Validation)**.** *Hazard analysis and mitigation claims should, if at all possible, be validated using mechanized/-computational techniques, and not just with subjective human judgment. And when modeled computationally, they should be validated with formal verification, not (only) with runtime assertion/property checking.*

Recommendation 52 (Security Mitigation Claim Validation)**.** *Security analyses and mitigations should, if at all possible, be validated using mechanized/-computational techniques, and not just with subjective human judgment. And when modeled computationally, they should be validated with formal verification, not (only) with runtime assertion/property checking.*

Too often, these kinds of analyses—risk, hazard, and security analysis and mitigation—are exclusively performed by experts using wisdom and checklists.

In our experience, when an existing subjective analysis (risk, security, correctness, and more) is augmented with mechanized reasoning, numerous flaws in the original analyses are discovered.

Therefore, for critical analyses like risk anlysis, we recommend that claims should be validatd using computational techniques. Doing so "supercharges" those very same experts with wisdom.

Recommendation 53 (Claim Fidelity)**.** *The* fidelity *of every claim must be explicit if it is not* precise*. A* precise *claim is one that is a theorem/property of model or code for which all assumptions are explicit for for which a proof could be created/generated. An* imprecise *claim is a claim that is not precise. The* fidelity *of a claim is the context of a claim and its evidence—particularly its set of assumptions—including any necessary caveats, error bars, or other statistical analysis of related measures.*

Classically, fidelity of measurement claims are characterized by statistical error bars, standard deviations and variance of measures and their distributions, etc. This kind of fidelity information is commonly wholly absent in MBSE artifacts which include simulation data.

Formal logical models usually only need a (formal) context statement in order to characterize fidelity: the set of axioms and assumptions made in writing, generating, or reasoning about the model suffices to explain how the model is valid, in what circumstances it may be invalid, and thus—by virtue of earlier recommendations on model property refinement—how those assumptions impact the model's implementation.

4.8 Key Questions to Answer

The final key questions to answer about a system developed with MBSE, augmenting all of the recommendations summarized above, are characterized by the following, final, recommendations.

Recommendation 54 (Understandable)**.** *An MBSE-based assurance case should be* understandable *to any moderately experienced engineer.*

Recommendation 55 (Believable)**.** *An MBSE-based assurance case should be* believable *to any moderately experienced engineer.*

Recommendation 56 (Coherent)**.** *An MBSE-based assurance case should be* coherent *to any moderately experienced engineer.*

Recommendation 57 (Surety)**.** *An MBSE-based assurance case should meet or surpass the* assurance threshold *set by a client (its* surety*), and the fact that an assurance case is not yet good enough should not be a surprise to any involved.*

Is a system and its assurance case fulfills these four recommendations—reinforced by demonstrated alignment with all of the earlier recommendations that align with an organization's review remit (be it validation, certification, statutory, or rule-based), then the system is fit for acceptance based upon MBSE artifacts.

Chapter 5

The HARDENS Reactor Trip System (RTS) Demonstrator

The HARDENS Reactor Trip System is a demonstrator model-based engineering system built for the NRC. It demonstrates the art of what is possible using modern model-based hardware, firmware, software, and systems engineering for safety-critical systems, using only freely available, mostly open source, technologies.

Chapter 3 and **??** review the state-of-the-art in commercial and freely available MBE technologies. In the conclusion of this document, chapter 6, we reflect upon how this demonstrator and its assurance case might different were proprietary tools used instead.

This chapter describes all of the modeling artifacts used in creation of the RTS, and grounds their use from a traditional MBE perspective, which has no rigorous or formal model-based assurance. As such, we differentiate the widespread state-of-practice against the state-of-the-art. We advocate that only the state-of-the-art is adequate to guarantee the correctness, safety, and security of nationally critical infrastructure.

After all, if Galois is able to design, build, and assure a system like the RTS for a nickle (relatively speaking), larger corporations should be able to do at least as well using the orders of magnitude greater resources that they regularly consume or spend.

1. This chapter is about describing the RTS from a software and hardware engineering perspective.

2. It describes modeling artifacts in the context of their use from a traditional MBE perspective, with no rigorous or formal model-based assurance.

3. Explain the RTS at a high level.

5.1 RTS Overview

The Reactor Trip System (RTS) is a demonstration of a digital instrumentation and controls (DI&C) system. As such, it features:

1. a sense-compute-control architecture;
2. a human-in-the-loop user interface;
3. a built-in self-test subsystem; and
4. a fault-tolerant design comprising components with identical behaviors implemented by multiple techniques.

In this chapter we discuss the artifacts comprising the RTS: not only the implementation artifacts, but also, crucially, the *assurance* artifacts.

5.2 Specifications

The RTS specification is naturally divided into a set of high-level system specifications comprising *requirements*, a *domain engineering model*, and a *feature model*. These high-level specifications are further refined by *formal* specifications of both the requirements and system architecture.

5.2.1 High-Level System Specifications

First, we will start with the NRC RFP as our first source for high-level system specifications. We will then review the relevant IEEE standard.

The Original NRC RFP. The core four pages of the original NRC RFP are included here in full on the following four pages. We will refer directly to elements therein in the following description of the **HARDENS Top-level Project Goals**.

In order to build our *domain engineering model* for the RTS as well as to make a start on our system requirements, we extract directly from this text, line by line, the following *goals* (section 5.3) and *scope of work* (section 5.3.1).

5.3 The Reactor Trip System's Goals

The goals of the are explicitly stated on the first page of the RFP. We manually highlight those declarations and transliterate them into a specification written in the LANDO system specification language (see appendix A).

Each goal has a *short name*, which we will use when referring to specific goals in what follows. In general, we label, rather than number, high-level specification artifacts such as goals, scope, and requirements in this fashion. Labels have more semantic meaning and permanence than numbers and, as these artifacts changes over time (e.g., a requirement changes mid-project), developers do not confuse themselves or the reviewer with missing, evolved, or struck-out numbered specification artifacts.

We call our demonstrator system the *Reactor Trip System*, or RTS for short.

Goal 1 (Project Focus)**.** *The project's main focus is on safety critical systems design.*

Goal 2 (Mitigate Complexity)**.** *The project must demonstrate MBSE's capability to address software and system complexity.*

Goal 3 (Enhance Productivity)**.** *The project must demonstrate MBSE's capability to address the productivity challenges of complex distributed embedded systems.*

Goal 4 (Regulatory Review)**.** *The project must demonstrate and help explain how Model-Based Systems Engineering (MBSE) methods and tools can support regulatory reviews of adequate design and design assurance.*

Goal 5 (Barriers and Gaps)**.** *The project must identify any barriers or gaps associated with MBSE in a regulatory review of Digital I&C for existing NPPs.*

Goal 6 (Superior for Complex SSCs)**.** *The project should demonstrate if MBSE, with its inherent ability to analyze and simulate many different scenarios, is a superior approach for more complex systems, structures, and components (SSCs).*

Goal 7 (Non-Document-based Evidence)**.** *NRC regulators must be prepared for MBSE applications where non-document-based evidence is a part of a safety evaluation.*

Goal 8 (Future Review Processes)**.** *The project should help the NRC understand if an MBSE-based assurance case presentation will help to enable the use of MBSE as an alternate review process DI&C-ISG-06, Rev. 2 (ML18269A259).*

5.3.1 Scope of Work

The scope of work for the RTS defines a framing condition on the system's requirements and implementation. Each scope declaration turns into a framing assumption or a constraint on the system, in its specifications, implementation, or assurance case.

Scope declarations often help tremendously with regards to constraining the space of possible design decisions and, when there are few-to-no hard constraints, also help create and evaluate a multi-dimensional objective function that helps a system's creators focus on finding the optimal product that solves a client's need.

Each scope definition below is, again, transliterated into LANDO. See appendix A.

Scope 1 (Simple Protection System)**.** *The demonstration implementation must be a simple protection system.*

Scope 2 (Modern Process)**.** *The demonstration implementation must be created using a highly integrated computer-based engineering development process.*

Scope 3 (MBSE)**.** *The demonstration implementation must be created using MBSE.*

Scope 4 (Module Functionality Completeness)**.** *All the modules of the simple protection system must be modeled functionally.*

[FPGA]

Scope 5. *One FPGA-based circuit card must be modeled/designed in detail.*

Scope 6. *The level of detail in the design and supporting analysis should address independence of functions.*

Scope 7. *Independence should address interfaces between functions.*

Scope 8. *Independence should address self-testing implemented on the circuit card.*

Scope 9. *Independence should address voting between protection system elements.*

Scope 10. *The final product must be the design itself and the associated evidence to demonstrate its technical soundness.*

Scope 11. *NRC technical staff for I&C must review the "demonstration" material and identify additional information needed or material that is not needed for regulatory purposes.*

Scope 12. *The research project is to explore the full scope of MBSE; in particular, it is not limited to simulation-based validation of I&C system designs.*

Scope 13. *The project should use models and simulations in the early design phases and for the validation of I&C system designs, such as to confirm required behavior and identify unwanted or undesirable interactions.*

Scope 14. *For each aspect of MBSE, the kind of engineering artifacts should be identified and described to include their use as evidence of the soundness of the design.*

5.3.2 Domain Engineering Model

Our *domain engineering model* (or just *domain model*, for short) is a description of those conceptual elements unique to the domain of discourse for a given system. One can (and we do) *extract* a domain model directly from the artifacts salient to a system: RFPs, client documentation, standards, statutes, peer-reviewed papers, and more.

Galois has developed an NLP extraction tool that consumes documents, such as files in the PDF, Word, and ASCII or Unicode text file formats, and produces a histogram of *concepts* characterized by *grammatical category*.

Using that corpus, we remove those concepts that are common, widely understood, or generic to computer science and mathematics, and we are left with a prioritized list of concepts that we must represent in order to understand and write specifications in a given domain. Those concepts (nouns), and their character (adjectives) and interrelationships (verbs and adverbs) are the domain engineering model.

The domain model provides the semantics for the system requirements. Another perspective is that we build our *vocabulary* for specifying the system via the domain model. The high-level domain model is specified in Lando. The RTS *domain engineering model* is represented at its highest abstraction level in a structured Lando specification, shown in fig. 5.5, which defines all of the RTS artifacts using natural language.

The relations at the bottom of the source indicate that the RTS system *contains* each of the subsystems defined earlier in the the document. Each of these subsystems is further elucidated, including identifying and defining the constituent components, in the relevant Lando documents.

The Lando model specifies the behavior of the RTS by defining a set of *events*, and using those events to define *scenarios*, which are (partial) executions of the system: the sum of the scenarios should define the legal behaviors of the system.

As an example, the external input actions of the system are specified in fig. 5.6. Generally, external events specify the different actions that the user can perform via the RTS user interface, including manually actuating a device or configuring various parameters such as setpoints or maintenance mode.

These events are used to specify scenarios, a selection of which appear in fig. 5.7. Scenarios are nothing more than a sequence of events. In section 5.8 we will see that these scenarios are used to perform validation.

5.3.3 Feature Model

The RTS system is actually a *family* of related systems. For example, the RTS can be configured to run (1) on a host system that simulates the behavior of hardware, or (2) on an *emulated* hardware platform, or (3) on a real device.

The specification of this *product line* is captured via a *feature model* specified in the LOBOT specification language. The model captures not only *variation points*, but also relations' valid values.

For example, note that the device can be configured, via `all_Devices_twins`, such that all devices are digital twins. A constraint (below the `where`) indicates that if this feature is set, then all devices (sensors and actuators) must have the corresponding twin feature set.

The LOBOT specification is heavily documented. Each *type* is used to capture a *feature* in the feature model. A *feature* is a decision (variation) point in the design of a product line. Legal/permitted sets of values of features are constrained by logical propositions included in `struct` declarations and their instances.

A `struct` is a collection of related features, listed as name-value pairs like in a typical *structure* found in many programming languages (what is known as a *labeled product type* in type theory), and a set of constraints which explain the legal configurations of its features. Constraints in LOBOT are specified after the *where* clause in the `struct` definition.

The LOBOT specification also acts as a specification for the build system, by precisely indicating which build configurations are available to the user. These constraints are expressed in the `where` clause of the `rts` struct. They specify that the `soc` must be a `Twin` exactly when all devices are twins, and the entire RTS runs as a virtualized platform exactly when either (a) the SoC is a twin, the board we are compiling to is a RISC-V VEGA board and there is no virtualized twin development twin platform in use (i.e., we are compiling to a real RISC-V CPU on a development board that is connected to sensors and actuators), or (b) the SoC is a twin and there is no board in use at all (thus, everything is running on a virtualized development platform, which as defined early in the specification, means that the RTS is running either on a POSIX host).

The LOBOT specification also includes a description of all legal products derivable from the RTS product line at this time. See the `rts_configs` instance of the `rts` struct and its associated constraints.

Finally, there are a small set of checks at the bottom of the LOBOT specification. These are theorems that the LOBOT tool will try to prove that must hold for all legal configurations of the feature model.

The first theorem, `check_bottom`, is a consistency check, akin to the requirements consistency checker provided in the FRET tool (section 5.3.4) and the consistency check we perform for the ACSL, CRYPTOL, and SAW specifications (cf appendices E.6 and E.9 and section 5.5.1). All of these "bottom" checks are discussed below in section 5.8.2.

The second, `check_twin_build_configs`, checks that all virtualized builds do not need a development board.

Besides reasoning about feature models, the LOBOT tool has many other features, including the ability to count the number of models that exist, iteratively enumerate models, and drive back-end tools to turn models into systems (e.g., BSV CPUs) and assurance artifacts (e.g., RISC-V compliance tests). We are not using any of those features in this project.

The full LOBOT specification of the RTS system is found in **??**.

5.3.4 Formalized Requirements

The *requirements* expressed in LANDO form the basis of the specification of the RTS. However, as they are written in natural language, they are not suitable for consumption by tools (such as formal verifiers) that can check whether or not a model (section 5.5) or implementation (section 5.7) satisfies a specification. The natural language requirements written in LANDO must be formalized before they can be used as specifications for tools "lower" in the modeling and implementation stack.

We use NASA's FRET tool to help produce formalized requirements from our LANDO specification. Use of a tool like FRET helps the designer in two complimentary ways.

First, FRET allows the user to use diagrams, simulators, and model checkers to explore requirement semantics. The user can use these tools to gain confidence that the formal representation accurately captures the intent of the corresponding natural language requirement. This analysis establishes the refinement step from the LANDO specification to its formalization.

Second, FRET can automatically machine-check requirements for *realizability*. Realizability-checking checks that an implementation of a component exists that conforms to the requirements given any combination of valid inputs. This analysis guarantees the requirements are non-vacuous, and can thus be used in further refinement steps.

Figure 5.9 is a screenshot of the FRET requirement editor. FRET's input format is a stylized form on English NASA calls FRETish. FRETish is fairly easy to write and read, especially with interactive tool support, has a formal semantics, and is machine interpreted.

In order to deeply understand and debug requirements captured in FRET, one can use the tool to automatically convert a requirement in FRETish into two different-but-equivalent semantic interpretations written in *Linear Temporal Logic* (LTL): *future time LTL* and *past time LTL*. Essentially, these interpretations are duals of each other, one explaining the meaning of a requirements with regards to the *past*, and the other with regards to the *future*.

Also, as mentioned previously, FRET permits one to reason about a set of requirements in several ways. Requirements can be checked for *consistency*, *completeness*, and *realizability*. All three checkers have a similar user interface. The user interface of the realizability checker is shown in fig. 5.11.

Every requirement specified in the top-level LANDO model which is about system properties is translated, using the RTS domain engineering model, into a

FRET requirement. Traceability is maintained using our standard methodology of name translation between models.

Requirements that are project goals, constraints/scope, or architectural are not translated into FRET, as they have no semantic meaning with regards to the architecture. The only way such informal/semi-formal requirements can be demonstrated is with informal/semi-formal assurance artifacts, such as the requirement *NRC Understanding*. The full set of project requirements are found in appendix A.9.

5.4 System Architecture and Models

The system model is refined from the LANDO specification to SYSMLV2. The SYSMLV2 models capture system-level aspects of the RTS. The full SYSMLV2 model is found in appendix D.

The structure of the SYSMLV2 specification includes a direct refinement of the LANDO specification. Structural properties are maintained through the refinement, all informal/semi-formal LANDO requirements are refined to SYSMLV2 requirements, the LOBOT model is refined to a SysML variation model, all formal FRET requirements are refined to SYSMLV2 properties, etc.

The SYSMLV2 specification, as it is a refinement of the LANDO specification, also includes new details about the system that were not expressed, or were expressed abstractly, in the LANDO specification.

For example, the hardware components used to build the demonstrator (e.g., all of the parts on the FPGA board, all sensors, solenoids, etc.) are itemized in the `RTS Hardware Artifacts` SYSMLV2 package. The hardware is assembled into the RTS demonstrator, as explained in the `RTS` part.

Additionally, the context and stakeholders that hold concern about the RTS system are also captured in the SYSMLV2 specification in the `RTS System Contexts` and `RTS Stakeholders` packages.

SYSMLV2 tools currently are only capable of type checking a SYSMLV2 specification and rendering views on that specification in SYSMLV2's graphical syntax. At the moment, there is no means by which to behaviorally simulate a SYSMLV2 dynamic model (such as an abstract state machine) or to refinement check that a SYSMLV2 model conforms to an abstraction (such as a LANDO specification) or a refinement (such as our CRYPTOL or ACSL models or their implementations). The development of such refinement tools is part of the focus on a new DARPA I2O project starting in late 2022 led by Galois.

5.5 Executable Behavioral Model

The next major refinement step of the RTS is an executable, denotational model of its behavior. In the RTS, our executable model not only serves as a reference to guide implementation, but is also used to *generate test vectors* (**??**), *synthesize*

implementations (section 5.7), and serve as a specification for formal tools to reason about implementation correctness (section 5.8).

5.5.1 Cryptol Model

The RTS executable model is specified in CRYPTOL. Each component is specified in its own CRYPTOL model.

For example, the `Actuator` model is shown in fig. 5.12. The model includes a definition of the `Actuator` state, which is a record consisting of the last input from the voting stages, plus the last manual actuation command: both are Boolean values (which is a `Bit` in CRYPTOL).

Next, the model defines three functions: two (`SetInput` and `SetManual`) which simply set one of the two input bits, and one (`ActuateActuator`) which defines the logic for combining the signals from the two `Actuation Logic` stages into a single actuator command (via a logic OR).

Part of the `Actuation Unit` is shown in fig. 5.13. The terms shown model the logic (`TemperatureLogic`) for whether or not a set of temperature readings should result in a trip signal generation. This model includes a (local) definition of 2-out-of-4 coincidence logic.

5.5.2 Formal Requirements Satisfaction

In both CRYPTOL models, we connect models to requirements to establish the refinement of the high-level specifications and requirements to the behavioral model.

Every FRET formal requirement is refined to a CRYPTOL property (what amounts to a first order theorem) and all demonstrated to hold on the CRYPTOL model using:

- dynamic checking through the use of CRYPTOL's `:check` command which randomly generates values and checks that the property holds for those concrete values, and
- formal verification, by *proving* that the properties hold for all inputs using a SAT or SMT solver, using CRYPTOL's `:sat` and `:prove` commands.

For example, in the `Actuator` module, we indicate that `ACTUATION_LOGIC_MANUAL_DEVICE_0` and `ACTUATION_LOGIC_MANUAL_DEVICE_1`, which capture the requirement that the actuators have a manual mode, is satisfied trivially by representing the manual actuation state explicitly.

In the listing from the `ActuationUnit` module, we show how we indicate the satisfaction of `ACTUATION_LOGIC_VOTE_TEMPERATURE`, which says that 2-out-of-4 coincidence logic should be used to determine if a trip signal should be generated. We capture this requirement as a *property*, which is just a function that tests whether or not the requirement is satisfied on a single input value. Properties such as these can be *proven correct* for *all inputs* using CRYPTOL and (using SMT solvers under the hood). In this case, proving the property correct means

that for all input signals, the model of the temperature logic returns a value consistent with the requirement.

In the full CRYPTOL model, which is located in appendix E, you will see dozens of properties about about the model, and all of them are formally verified (see the `check_model.icry` script which is driven by the model Makefiles's `check` rule) and can dynamically validated, if one so desires.

5.6 Behavioral Model-based Interface Specification

Our CRYPTOL model is next refined to an ACSL specification of a C architecture. The C architecture is specified by refining the SYSMLV2 and CRYPTOL models, preserving structure and properties as normally done in our methodology.

Such a refinement not only makes for simple traceability up and down the refinement hierarchy, but also helps make formal verification of the implementation (both software and hardware) straightforward, as discussed below in section 5.8, such as in the use of SAW for proofs of correctness of software components against their CRYPTOL specifications.

ACSL is the de facto standard formal specification language for C and C++ programming and is specified in an ISO standard. Many tools understand ACSL, most of which are a part of the Frama-C tool suite from CEA. ACSL specification are written either in C header files, as seen in the RTS's `instrumentation.h` and `platform.h` files found in appendix E.9, or in C implementation files, as seen in the `instrumentation.c` and `actuation_unit.c` files, all of which are included in appendices E.7 and F.

5.7 Implementation

The RTS implementation is a refinement of its higher-level LANDO, SYSMLV2, and CRYPTOL specifications. Every component of the implementation—including the hardware and bare-metal software—is derived directly from earlier specifications.

5.7.1 Hardware Components

The RTS hardware consists of a small System-on-Chip that contains one or three instances of a small RISC-V RV32 CPU called the NERV CPU, developed by our colleagues at YosysHQ.

NERV stands for the **N**aive **E**ducational **R**ISC-**V** Processor. It is a simple, small, open, RISC-V CPU that has been formally assured for correctness against the RISC-V ISA specification. We chose to use NERV exactly because it had these properties.

Overall Implementation Size, Shape, and Location. The overall implementation consists of:

- a set of Bluespec SystemVerilog (BSV) modules that describe our RISC-V System-on-Chip, spread across ten files located in hardware/SoC/src_BSV,
- a set of Verilog files which are compiled from the aforementioned BSV (six files located in hardware/SoC/verilog),
- a set of Verilog files imported in order to implement the SoC, either in simulation or emulation, located in hardware/SoC/src_Verilog and hardware/SoC/Verilator_RTL, and
- a set of SystemVerilog files (three generated files located in src/generated/SystemVerilog, and one that is handwritten located in src/handwritten/SystemVerilog).

The Bluespec SystemVerilog is just over 1,000 lines long. The Verilog and SystemVerilog is only around 150 lines long.

RISC-V. RISC-V is the Instruction Set Architecture (ISA) for the fifth generation of RISC processors. It is an open, unencumbered ISA, therefore anyone can use the ISA for any purpose with no cost, and no encumbrance, to design or manufacture FPGA or ASIC-based CPUs. Dozens, and perhaps now, hundreds of RISC-V CPUs have been developed over the past eight years.

The software ecosystem that sits on top of RISC-V is very rich as well, and includes operating systems (Linux, FreeBSD, FreeRTOS, and others), compilers (gcc and clang both fully support RISC-V), and much more.

The RISC-V ISA is semi-formally specified via a set of standards documents and is formally described in a specification in the SAIL language/platform.

The NERV CPU is formally verified against an interpretation of this specification using the `riscv-formal` assurance framework, also developed by YosysHQ. Using `riscv-formal`, one can prove, using a hardware formal verification tool like JasperGold, OneSpin, or Yosys, that every instruction in the ISA is exactly implemented correctly according to the specification.

The simple, clean, elegant, precise design of the RISC-V ISA is what makes this level of assurance possible. One cannot, in general, formally assured any other flavor of CPU (x86, ARM, MIPS, etc.) against its ISA as easily, or even at all.

The NERV design is located in the HARDENS/hardware/nerv directory, which is, strictly speaking, just a submodule of our project directly imported from YosysHQ's GitHub project.

The RTS System-on-Chip. The RTS SoC is a small, one or three core, CPU. Its design is exactly driven by the requirements that flow down to the hardware via refinement through the specifications previously summarized.

The design of the RTS SoC is located in the HARDENS/hardware/SoC directory[1]. The SoC is written largely in the Bluespec SystemVerilog (BSV) language, which is a high-level Hardware Description Language (HDL) which compiles to the Verilog HDL. The Bluespec tool suite includes a compiler called `bsc` which can compile BSV to either an executable software emulator of the design or a Verilog RTL implementation. Either can execute the RTS system, though we only use the latter in the project.

Simulating or Emulating the CPU or SoC. In order to execute a Verilog design, such as the RTS CPU or SoC, one use any number of the following choices:

1. simulate in a commercial/proprietary RTL simulator, such as those available from Cadence, Synopsys, or Siemens,
2. compile to a software simulator using a complementary commercial/proprietary tool from one of those three EDA firms or a handful of others (such as Metrics),
3. compile to a software simulator using an open source, freely available RTL compiler such iVerilog, Verilator, or Yosys,
4. compile to a hardware FGPA-based emulator using a commercial/proprietary tool chain such as those provided by Xilinx, Intel, Lattice Semiconductor, and others, or
5. compile to an open source FGPA-based emulator using an open source, freely available tool chain such as that provided by YosysHQ, Berkeley's FireSim, or others.

For the HARDENS project, we use the Verilator simulator and the YosysHQ and Lattice Semiconductor-based FPGA compilers.

SoC Design. The SoC is either one or three core because the RTS software implementation can either run on a single CPU core or can be split across three CPU cores. See section 5.7.2 for more information.

The three CPU design is novel because it closely adheres to the requirements that refine from the system and model-based designs.

Traditional architectures for a three core CPU would either have (a) three completely separate CPUs that have separate memories and communicate via message passing or monitors, or (b) three CPUs that share a single memory and a level two cache in order to communicate via more traditional, ad hoc shared memory memory mechanisms.

For the RTS SoC, we find that only two of the CPUs need to communicate with each in one very small fashion, discussed below. In particular, we arrange the RTS architecture on the hardware in the following way:

[1] Located on GitHub at HARDENS/hardware/SoC

- On CPU #1, we place computation that communicates with the outside world over a UART, both for programmability and the RTS user interface.
- On CPU #2, we put instrumentation and actuation communication and computation in order to read sensors, drive actuators, and make instrumentation decisions.
- Finally, on CPU #3, we put the core abstract state machine that is the core computational loop of the RTS.

Product Configurations. Based upon our architecture, the RTS can be deployed in one of several different product configurations. The entire system can be deployed in one of three ways:

1. On a single FPGA, which emulates either a single CPU or a three core CPU.
2. On three FPGAs, each of which is a single independent CPU, and information is transmitted between the two CPUs which must communicate via a simple mechanism like a GPIO line.
3. One three normal compute elements, such as a normal Linux machine or a computational container (such as a Docker or Kubernetes instance), each of which either simulates a RISC-V CPU or simply runs the POSIX version of the subsystem. In such a configuration communiation betwen compute elements would take place over, e.g., Ethernet.

The current implementation has much of what is necessary to realize any of these configurations, but the only configuration that is complete and delivered is the POSIX digital twin configuration and much, but not all, of the single FPGA configuration with the three core SoC.

Hardware Divisions of Instrumentation. The divisions of instrumentation are redundantly implemented, as described in the proposal and the system architecture, as two software and two hardware implementations, and one of hardware implementations is implemented by hand and the other is automatically generated from our Cryptol model (we discussed generation below in section 5.7.3).

Because the instrumentation is implemented in hardware, as are some of the drivers for the sensors and actuators, for that matter, the FPGA will execute those modules concurrently, independently, and if one of them were to fail for some reason (e.g., were it to deadlock), it will not affect the execution of the other modules.

The source for these two implementations is located in appendix E.7 and in the project filesystem in src/generated/SystemVerilog and src/handwritten/SystemVerilog. These implementations are demonstrated to be functionally equivalent to each other through formal verification, as discussed below in section 5.8.

Current State of the Hardware Implementation. Despite all of the work on formal assurance and system modeling, the hardware implementation is not complete. In particular, we are 90% of the way to completing synthesis and debugging the three core RISC-V SoC, but it (a) does not operate properly, and (b) we are not yet cross-compiling the RTS system to deploy them on the SoC. The reason that we have not finished the hardware implementation is that we have run out of project resources, given the very small budget for the project, and the remaining work is purely engineering, and involves no formal assurance or novel research. If more resources are made available on this topic, we can easily finish up the implementation.

5.7.2 Software Components

The software components in the RTS implementation are arranged into 8 header files and 23 implementation files, all of which are implemented in verifiable C. The total size of the implementation is approximately 2,300 lines of code, and that includes all assurance-related code as well.

The actual size of the core behavioral implementation of the RTS (including all product variants and digital twins) is only 1,400 lines of hand-written C code and 700 lines of automatically generated C code.

Header Files. All header files are located in src/include. The header files contain type declarations for named C types, functions, and ACSL behavioral specifications of both. The core ACSL behavorial axiomatic specifications are located in src/include/models.acsl. We discuss these model-based ACSL specifications below in section 5.8.3.

All common types and constants are located in src/include/common.h. Every single type declaration and constant is refined directly from the system architecture and CRYPTOL model.

Product line variants are concretized in three macros at the bottom of `common.h` called `VARIANT`, `VARIANT_IMPL`, and `VARIANT_IMPL2`.

There are a handful of helper macros defined to help write terse specifications. They are:

- `ShouldTrip` in src/include/instrumentation.h,
- `DEBUG_PRINTF` in src/include/platform.h,
- `ASSERT` in src/include/platform.h, and
- `MUTEX_LOCK` and `MUTEX_UNLOCK` in src/include/platform.h.

Implementation Files. The software implementation is written to be portable across any typical POSIX build platform and as a baremetal implementation on an arbitrary ISA. Each device in the instrumentation subsystem is implemented as both a software digital twin (a mock of our real hardware) or as a software

driver to the real hardware. (Recall from the discussion above in section 5.7.1 that each device is also implemented as a hardware driver in parallel.)

Some of the implementation files are used to provide assurance of the low level model(s) of the RTS or its implementation. For example, src/bottom.c is used to formally verify model consistency, and runtime tests are located in the files in the src/tests directory. We discuss their design and use below in section 5.8.

The implementation, as discussed above in section 5.3.3, has several variants. Code that is used to generate implementations of these variants is located in src/variants, and basically amount to setting a handful of variation point constants from the LOBOT model to the right values for the implementation and using `#include` to import the variant code.

Code used to generate model-based tests and run bisimulation is located in src/self_test_data and is discussed below in section 5.8 as well.

The two top-level files that represent the core of the two main variants of the RTS (POSIX and baremetal) are located in src/posix_main.c and src/rv32_main.c. The former implements both a single-threaded and concurrent version of the software digital twin. The feature that controls which variant is built is called `USE_PTHREADS`. The latter has to include variant BSP-specific header files, I/O definitions, and more, typical of any embedded systems code.[2]

5.7.3 Model-derived Components

Two kinds of RTS implementation components are automatically generated from models. Both are critical parts of the instrumentation subsystem, and as described in the original HARDENS proposal, nearly all are a *pair* of implementations of different components of instrumentation—the actuation unit, the actuator, the saturation check, and the instrumentation component—that are meant to be exactly behaviorally equivalent. A discussion of the formal evidence for such is included below in section 5.8.

Some of the assurance artifacts are also automatically generated from models, and are thus model-derived, but they are not discussed here.

In general, each automatically generated software component is created by the CRYPTOL tool, which has the ability to compile executable CRYPTOL models into correct-by-construction implementations in C, JVM, and SystemVerilog. We use two of these three backends; the software-targeting model compiler is called `crymp` and the hardware-targeting model compiler is called `cryptol-verilog`. The SAW tool, which we use in this project for formal assurance of some of the software implementation, also has the ability to compile to correct-by-construction implementations as well (in LLVM).

The `generate_sources` build rule and its dependencies in the main software Makefile is how code generation is controlled. It is decomposed into two rules, one to build software (`generate_c` and its dependencies) and one to build hardware (`generate_sv`).

[2] BSP means Board Support Package.

5.7.4 Hand-Written Components

The hand-written components of the RTS software stack include:

- the trivial "bottom" implementation that is used to check CRYPTOL and ACSL specification soundness/consistency (discussed below in section 5.8),
- the software common across all variants, including core data type declarations, functions which interoperate with instrumentation subsystems (reading channels, device values, trips, modes, state, resets, self-testing, etc.), all of which are located in src/common.c,
- the RTS UI, located in src/core.c,
- the RTS self-test functions, located in src/core.c,
- the wrapper around the four different-but-equivalent critical intrumentation subsystems, located in src/sense_actuate.c, and
- the core abstract state machine states and transitions of the RTS, implemented twice for the two main product variants, as discussed above, located in src/posix_main.c and src/rv32_main.c.

Tracing and Understanding Software Implementations and Their Specifications in the RTS. In general, every implementation component—whether it is a datatype declarations, a store reference declaration (that is to say, a memory location that is named, typed, in a fixed location, and holds a particular piece of RTS state), or a function—is specified semi-formally in LANDO (in English) and formally (in ACSL and CRYPTOL). One can trace between these elements using our name-based, refinement-centric traceability methodology.

In this document's appendices (**??**–**??** we provide an explicit means by which to trace between all components and subsystems. In what follows we provide some examples of this traceability.

Reviewing and Tracing Software. One can discover where the core software located in src/common.c is documented by following its dependencies. For a C implementation, one looks at the (transitive) set of header files that are included via C preprocessor `#include` directives. For example, in the case of `common.c`, these files are `core.h`, `platform.h`, `actuate.h`, and `rts.h`. Likewise, `core.h`, in turn, includes `common.h`.

These dependencies are depicted in **??**. Note how header file dependencies largely mirror package dependencies in the system architecture.

Each type declaration—both data types (such as the various array types in `platform.h` and structure types in `common.h`) and function types (such as those found in `actuation_logic.h`) in the C code refine isomorphic composite and functional data types, respectively, in either or both the ACSL and CRYPTOL specifications. Likewise, each function further refines those defined in the LANDO specification or the same name.

For example, the Mission Essential Function (MEP) C function `Is_Ch_Tripped`'s C type signature is defined in line 37 of `instrumentation.h`, its meaning is defined in the associated contract on line 35, in the ACSL model in line 49 of `models.acsl`, which states on line 48 that it refines the CRYPTOL specification `RTS::InstrumentationUnit::Is_Ch_Tripped` which is located on line 138 of `InstrumentationUnit.cr`, which in turn refines the SysML specification starting on line 146 of `RTS_Static_Architecture.sysml`, which in turn refines the LANDO specification found in `instrumentation.lando`.

Likewise, you will find alternative implementations of `Is_Ch_Tripped` in hardware in the Verilog files `mkInstrumentationHandwritten.v` and `mkInstrumentation-Generated.v`, the Bluespec SystemVerilog files `Instrumentation_Generated_-BVI.bsv` and `Instrumentation_Handwritten_BVI.bsv`, the Verilog generated from CRYPTOL in `Is_Ch_Tripped.sv` and its hand-written sister in `Is_Ch_-Tripped_Handwritten.sv`, etc.

5.8 V & V artifacts

5.8.1 Reviewing and Tracing Evidence

To continue with the traceability discussion of the last section, one can trace from source code and specifications to all of the related validation and verification by continuing to follow names in validation and verification files.

So, in the case of `Is_Ch_Tripped`, we can find hardware formal verification artifacts in `saw/generated/Is_Ch_Tripped.yosys` and `saw/handwritten/Is_Ch_-Tripped.yosys`, and on lines 10, 11, 36, and 37 of `saw/instrumentation.saw`.

5.8.2 Specification Consistency

The first critical property to check before anything other is *specification consistency*. A specification is consistent if it contains no logical contradictions; occasions where both a proposition ϕ and ϕ are stated to hold simultaneously.

If any fragment of specification contains an inconsistency, and a verification depends upon that fragment, then the inherent contradiction implies that the precedent of one or more theorems we intend to prove is false, and thus the those theorems trivially hold, since the proposition $false \rightarrow B$ holds for any B.

Such specification inconsistencies are the bane of formal verification practitioners the world-over. When a hidden inconsistency happens, usually through a specification error, suddenly automatic verification becomes very simple, and even the hardest verifications go through immediately. This is a warning sign, and the consistency problem must be addressed.

In order to avoid this circumstance, the methodological mitigation we use in RDE is to always state false theorems (what are called "bottoms") about every goal, requirement, model, and implementation, and try to prove those false theorems are valid (true). If we can ever prove such a fallacy, we know our specifications are inconsistent, and must be corrected.

We thus construct "bottoms" for every layer of our RDE specification. In the case of HARDENS, we must do so in our formal requirements specification (in FRET), in our feature model (in LOBOT), in our denotational and executable model-based specification (in CRYPTOL), in our proof scripts (in SAW), and in our model-based behavioral interface specification language (in ACSL).

Our verification bench must check, preferably automatically, each of these "bottom" theorems, prior to attempting to check any other evidence: any "real" theorem about the system.

For our formal requirements specification, the FRET tool includes a specification consistency checker that does just that. In our LOBOT specification, the `check_bottom` property on line 242 of `RTS.lobot` is this theorem. For our ACSL specification, the "bottom" C implementations of every core function of the system, located in `bottom.c`, does just that too.

Once all consistency checks pass, we can now focus on using specifications to check the correctness of the RTS.

5.8.3 Model-based Assurance

Correctness checking is based upon *evidence artifacts* which demonstrate that *properties* which imply *requirements* and their associated *goals* are satisfied.

Evidence comes in several forms, ranging from semi-formal and manual to formal and automatic. We demonstrate the use and presentation of this full range of evidence artifacts in the assurance of the RTS.

Historically, especially for document-based assurance methods with manual review of assurance cases, as done in a handful of national and international certification standards—such as the NRC's, the FAA's DO-178, Common Criteria, DoD branch flight certification, and the NSA's "old" style certification—only semi-formal arguments written in natural language (i.e., English) are under evaluation.

While those arguments often call out to, or summarize, external technical evidence artifacts, such as tests or third party peer-review, they do not often refer to formal models, source code, or their properties. With model-based assurance, new forms of evidence become not only possible, but reasonable, as we can now make direct reference to formal models, source code, and their properties.

This evolution is comparable to the difference between an informal argument which appeals to common sense and intution about statistical matters and a formal argument that *uses statistics* to prove a supposition. Likewise, it is the difference between a middle school math book that teaches by example and rote and a university text book which teaches and uses theorem proving techniques.

In computer science and mathematics (especially logic), a similar transition has been underway over the past twenty years, elevating precision and evidence even from this latter example. Mathematics textbooks and top computer science conference papers used to be written only in very precise English—complete with formal definitions, theorems, and proofs (aka formal models and their properties)—until a movement started in *open mechanization*.

Mechanization is the process of making a formal, human-readable, human-writable, human-reasonable model and its properties machine-readable, writable, and reasonable. By "reasonable" we mean the steps taken in making or evaluating a model, a property/theorem, or a proof. Moving from (only) human-based to both human- and machine-based means that a computer double-checks our work. It also means though that we cannot elide, skip steps, or only appeal to a reader's intuition. Every single aspect of a model and proof is laid bare to both the human and the computer.

Mechanization does not mean that the models and proofs are **not** human-readable or approachable. The tools that are used for these kinds of model-based specifications and arguments are largely designed by and for mathematicians/-logicians, after all.

The tools that we use in the creation of the RTS demonstrator are a subset of the tools widely available for mechanized model-based systems engineering. These tools and others are discussed broadly, and in more detail, in **??**. The tools we chose to use in HARDENS are those we deem appropriate for the task: they are some of the best tools available, they lean into the kinds of modeling, properties, and evidence necessary for nationally critical infrastructure, and they are open source and freely available.

In what follows we move through each kind of model and evidence present in the RTS. By discussing each flavor separately, we can both demonstrate that the RTS is high-assurance and fulfills its semi-formal requirements and also teach about best practices in the presentation and evaluation of model-based systems.

5.8.4 Kinds of Evidence

To recap our discussion of evidence from earlier chapters, the new kinds of evidence that are available via model-based engineering largely fall into four categories: (1) the *application of rigorous patterns of relations between artifacts*, (2) *dynamic demonstration of property validity* against models and programs, (3) *static demonstration of property validity via automated solving*, and (4) *static demonstration of property validity via mathematical proof*.

While each of these forms of evidence is present in the RTS in plurality, we only discuss one or two examples per flavor so as to not be overly pedantic and overwhelm the reader. Given our recommendations in chapter 4, especially those in section 4.3, it should be unnecessary to pedantically discuss in an informal fashion, as we do here, every nook and cranny of every model, property, and piece of evidence.

Instead, a reviewer should be able to find the rhythm of a model-based system's design, implementation, and assurance and, by judicious randomized exploration of artifacts, quickly gain surity or distrust of a system and its assurance. By demanding less from both sides of the reviewer/reviewee relationship, by embedding more in models, the creation, presentation, and review of evidence is significantly less costly and time-consuming that traditional, pedantic-but-informal, processes.

5.8.5 Assurance of the RTS

In this section, we move through each layer of the RTS specification and assurance case, from most informal to most formal, following traceable lines of evidence and refinements.

In the RTS implementation, the file `Assurance.md` also contains a detailed write-up of the rigorous assurance of the entire system.

1. test vectors for self-test implementation
2. equivalence proofs of cryptol & implementation (SAW)
3. ACSL model refinement of cryptol (in-line with source)

From the Lando Specification. As discussed in section 5.3, the LANDO specification contains the transliteration of RFP and IEEE goals and requirements for the RTS. Thus our relations and refinements start with precise English properties expressed in LANDO, and refine to specifications in our feature model (in LOBOT), formal requirements (in FRET), and our system architecture (in SYSMLV2).

Some of these requirements and goals are extremely informal, such as **??**:

Evidence for our fulfilling these kinds of requirements is provided only through reference to concrete artifacts, such as project contracts, this report, project communications, slide decks that summarize the material, etc.

Other system or project requirements, especially ones that are semi-formal and can be refined to formal models, are liked in the assurance case to their refinements. For example, the requirement **??** points directly to the system architecture and its software and hardware:

Likewise, several requirements have a formal meaning and are directly traced to formal assurance artifacts, especially in FRET, and hence CRYPTOL, models. A good example of such a requirement is **??**, which refines to several different specification and assurance artifacts:

Recall that structural models refine directly to models with refined structure. For example, the LANDO system architecture, instrumentation, and dataflow specifications have this nature, and all directly refine to naturally named structures in other specifications.

From the Lobot Specification. The feature model specification written in LOBOT describes the products derivable from the product line. Its refinement connects to the build system that is used to generate products and to architectural, specification, and code components that help concretize those products.

The RTS build system is found in a set of simple `Makefile`s. Each build file attends to a separate concern, based upon its location in the filesystem.

- hardware/SoC/firmware/Makefile builds the RTS firmware for the FPGA-based demontrator.
- hardware/SoC/Makefile builds the RTS SoC and its digital twins.

- hardware/nerv/Makefile builds the NERV's digital twin (using iVerilog), basic firmware, and testbench.
- saw/Makefile compiles critical components of the RTS using the LLVM compiler to LLVM bitcode and runs SAW to formally demonstrate assurance of those components.
- specs/Makefile runs the LANDO and LOBOT tools to check all of the high level specifications.
- Makefile runs ...
- models/Makefile runs CRYPTOL to assure all properties to validate (via 10,000 model-based randomly generated tests) and formally verify the CRYPTOL model of the RTS.
- report/Makefile runs the `latexmk` tool to build the RTS final report (this document).
- src/Makefile builds the RTS "bottom" implementation and all digital twins, generates all model-based test data from CRYPTOL, builds all variants, runs various tools to automatically generate model-based implementations (software and hardware), and the hardware digital twin using `verilator`.

From the FRET Specification. The FRET specification is encoded in the filesystem using an internal JSON format supported by the FRET tool. That specification is flattened into a human-readable model of all of the RTSFRETish requirements in appendix C.

Each FRET requirement is refined from a SYSMLV2 requirement, which in turn is refined from a LANDO requirement. Refinement of semi-formal natural language specifications, such as requirements, means that either the text of a given requirement is syntactically identical on both ends of the relation (i.e., the text in the LANDO is character-by-character equal to that which is in the SYSMLV2), or the refined requirement has more information.

From the SysMLv2 Specification. The main properties *uniquely* described in the SysML specification is the structure of the RTS system. Consequently, the main assurance artifact associated with this specification is solely the structure-preserving refinement that is used upward (abstractly) to the LANDO specification, and downward (concretely) to the CRYPTOL and ACSL specifications described below.

A full, rich SysML specification would also include system properties (such as those captured in our FRET specification) as expressed in the SysML model. But those are wholly redundant in the RDE methodology, as they would be nothing more than a transliteration of the syntax of FRET to the syntax of SysML expressions. Thus, they are elided.

From the Cryptol Specification. The CRYPTOL specification, as discussed earlier, contains a transliteration of every FRET requirements and every behavioral property of the system as CRYPTOL first-order properties.

CRYPTOL properties are *theorems* about the CRYPTOL model. Thus, the assurance of the CRYPTOL model is wholly dependent upon the completeness, consistency, and correctness of these theorems.

We demonstrate the assurance of the CRYPTOL model by runtime checking and formally proving every property on the Cryptol model. The assurance script located at `models/check_model.icry` loads in the specification of every critical subsystem of the RTS model (e.g., `:m RTS::InstrumentationUnit`) and either `check`s or `prove`s all properties therein.

Recall that every FRET property is transliterated to a CRYPTOL property. Consequently, proving every property of the model proves that the model is complete with regards to the properties (and thus the original system requirements), and is consistent (because we cannot prove the false theorem), and is correct (because the properties hold for any realization of the system under all inputs, given we are largely using the `prove` command everywhere.

The final property, `end_to_end_test` is a property that specifies how the whole of the RTS holds together. As it is a trace property through the entire system, it cannot be automatically formally proven, but only simulated and runtime verified.

From the ACSL Specification. Key properties of critical components from the aforementioned models are also transliterated, if relevant, into the ACSL model of the software system.

Properties becomes assertions of various kinds—preconditions, postconditions, invariants, and assertions—and thus express the correct behavior of the system at the software refinement level.

We use these models to formally assure the correctness of the hand-written and automatically generated C implementations of these critical components (e.g., those in `instrumentation.h`). Several kinds of analysis are used, in series, to build up to a full formal assurance case. First we prove that the code contains no underdefined behavior and then we perform full formal verification of the implementations against the contracts.

Normally we would also use these models to automatically generate a model-based runtime verification harness for the software implementation so as to demonstrate that all checks are valid as runtime assertion checks on all digital twins and on the deployment platform. This project had insufficient resources for us to also add this extra level of assurance.

From the SAW Specification. For C implementations, we use SAW to prove correctness between the Cryptol model and the LLVM bitcode. For SystemVerilog implementations, we use SAW to extract a Verilog module from the Cryptol specification, and then use the Yosys tool to prove equivalence between

the extracted Verilog module and the generated or hand-written SystemVerilog module.

Not only does this establish the functional correctness of the component in question, but writing such specification (that can be precisely checked) ensures that the interface between components is made precise. For example, the final specification must take into account any differences in endianness assumptions between the source (Cryptol) and the target (e.g., the Verilog hardware design).

5.8.6 Hand-written

We use a tool called `pexpect` to do end-to-end testing of scenarios. Each test case is a transcription of a LANDO scenario that matches UI output. The file `tests/README.md` explains the end-to-end runtime verification for the RTS in detail.

Each test scenario specified in `specs/test_scenarios.lando` is translated into a parametrized software test, either driven directly against the software through function call invocation or, better yet, as driven through the top-level UI.

By driving through the external UI, we can use the same model-based parametrized tests to drive digital twins (like the CRYPTOL model or the software simulation of the RTS in either the `Posix` or `RV32_bare_metal` configurations of the product line) or directly on the FPGA deployment platform.

Title: Assessment of Model-Based Systems Engineering Processes in a Regulatory Review Context for Digital Instrumentation and Controls of Existing Nuclear Power Plants

C.1 Background

Over the past 15 years, Model-Based Systems Engineering (MBSE) has emerged as powerful methodology and practice for realizing and verifying complex embedded systems while providing rigorous evidence of functional and safety compliance. The phrase "Model Based Systems Engineering" has been used in many different contexts to the point where its meaning and purpose is vague. For safety critical systems design, the definition from the systems engineering and the formal methods community is appropriate.
"Model-Based Systems Engineering (MBSE) is the formalized application of modeling to support system requirements, design, analysis, verification and validation activities beginning in the conceptual design phase and continuing throughout development and later life cycle."
Because of its capability to address the software and system complexity and productivity challenges of complex distributed embedded systems, MBSE is quickly becoming the preferred engineering paradigm for the development of such systems across a variety of application domains. However, MBSE has not been used in any significant degree in the nuclear industry.

The NRC's guidance for digital I&C has its origins in a document-based waterfall development model. Modern software (SW) engineering environments are increasingly using highly integrated ecosystems of SW and hardware (HW) development tools with less reliance on published documents during development. These approaches tend to view the model as a portable "executable specification." The NRC is investigating the potential utility and impacts of MBSE methods and tools to the established review process of I&C for nuclear power plants (NPPs).
Documents are created by and for people to use (in the nuclear I&C domain) and are mostly natural language descriptions. Electronic representations, such as models, can be statically and dynamically analyzed by electronic tools. Although a human can never be removed from the development process, certain analysis tasks can be performed more quickly and accurately by electronic tools. Such tools may be difficult to qualify using current NRC criteria, which is focused on whether defects in the tool or resulting software would not be detected by other verification and validation activities.

C.2 Objective

The objective of this contract/order is to obtain expert technical services in order to develop a better understanding of: (1) how Model-Based Systems Engineering (MBSE) methods and tools can support regulatory reviews of adequate design and design assurance, (2) identify any barriers or gaps associated with MBSE in a regulatory review of Digital I&C for existing NPPs.

C.3 Scope of Work

The proposed research approach is for the implementation of a simple protection system using both: (1) highly integrated computer-based engineering development processes, and (2) MBSE. All the modules of the simple protection system would be modeled functionally, and one FPGA-based circuit card would be modeled/designed in detail. The level of detail in the design

Figure 5.1: NRC RFP Page 5

and supporting analysis should address independence of functions. Independence should address interfaces between functions and self-testing implemented on the circuit card, as well as voting between protection system elements.

The final product would be the design itself and the associated evidence to demonstrate its technical soundness. Then the NRC technical staff for I&C would review the "demonstration" material and identify additional information needed or material that is not needed for regulatory purposes.

Different parties have different ideas about what it means to use MBSE. This research is intended to identify and explore an existing state of the practice, and not to develop new engineering practices.

MBSE, with its inherent ability to analyze and simulate many different scenarios may be a superior approach for more complex systems, structures, and components (SSCs). Modern digital systems tend to be more complex because they are SW intensive and include more shared resources, more coupling of resources (e.g., digital communication, heterogeneous computing devices), and sometimes adaptive abilities. For these reasons, MBSE is often needed to design such systems to high levels of design assurance. Therefore, the regulator must be prepared for MBSE applications where non-document-based evidences are part of a safety evaluation.

For relatively simple applications, such as a reactor trip system, MBSE may support more robust analysis methods such as formal methods or model-based safety assurance.

This research is to explore the full scope of MBSE (i.e., it is not limited to simulation-based validation of I&C system designs). The use of models and simulations in the early design phases of new NPPs and validation of I&C system designs (e.g., confirm required behavior and identify unwanted or undesirable interactions) is one aspect of MBSE. For each aspect of MBSE, the kind of engineering artifacts should be identified and described to include their use as evidence of the soundness of the design. The alternate review process of DI&C-ISG-06, Rev. 2 (ML18269A259), could help to enable the use of MBSE.

Base System Architecture:

- Four redundant divisions of instrumentation, each containing identical designs:
  - Two instrumentation channels (Pressure and Temperature)
    - Sensor
    - Data acquisition and filtering
    - Setpoint comparison for trip generation
    - Trip output signal generation
- Two trains of actuation logic, each containing identical designs:
  - Two-out-of-four coincidence logic of like trip signals
  - Logic to actuate a first device based on an OR of two instrumentation coincidence signals
  - Logic to actuate a second device based on the remaining instrumentation coincidence signal

Functions to Be Implemented:

1. Trip on high pressure (sensor to actuation)

Figure 5.2: NRC RFP Page 6

2. Trip on high temperature (sensor to actuation)
3. Trip on low saturation margin (sensors to actuation)
4. Vote on like trips using two-out-of-four coincidence
5. Automatically actuate devices
6. Manually actuate each device
7. Select mutually exclusive maintenance and normal operating modes on a per division basis
8. Perform setpoint adjustment in maintenance mode
9. Configure the system in maintenance mode to bypass an instrument channel (prevent it from generating a corresponding active trip output)
10. Configure the system in maintenance mode to force an instrument channel to an active trip output state
11. Display pressure, temperature and saturation margin
12. Display each trip output signal state
13. Display indication of each channel in bypass
14. Periodic continual self-test of safety signal path (e.g., overlapping from sensor input to actuation output)

Characteristics to be demonstrated (based on the requirements of IEEE Std 603-2018):
1. Completeness and consistency of requirements
2. Independence among the four divisions of instrumentation (inability for the behavior of one division to interfere or adversely affect the performance of another)
3. Independence among the two instrumentation channels within a division (inability for the behavior of one channel to interfere or adversely affect the performance of another)
4. Independence among the two trains of actuation logic (inability for the behavior of one train to interfere or adversely affect the performance another)
5. Completion of actuation whenever coincidence logic is satisfied or manual actuation is initiated
6. Independence between periodic self-test functions and trip functions (inability for the behavior of the self-testing to interfere or adversely affect the trip functions)

Task 1:
The Contractor shall implement the system described above using both: (1) highly integrated computer-based engineering development processes, and (2) MBSE. All the modules of the simple protection system would be modeled functionally, and one FPGA-based circuit card would be modeled/designed in detail. The deliverable will be the model-based design itself.

Task 2:
The Contractor shall perform preliminary V&V and testing of the design using model-based engineering and testing methods. The deliverable will be the artifacts as described in the proposal.

Task 3:
The Contractor shall participate in an evaluation of the artifacts produced in tasks 1 and 2 with NRC staff. This will consist of:
1. An initial kickoff meeting for this task with the NRC staff.
2. The NRC staff will then provide initial feedback on the artifacts produced and additional information that would be needed.
3. The Contractor shall then attempt to address any issues and provide the additional information within the time for this portion of task 3 (1 month).

Figure 5.3: NRC RFP Page 7

4. A second meeting will be held to discuss the additional information provided.

Task 4:
The Contractor shall develop a one-day long virtual presentation on the results of this research that explains the MBSE approach used, the engineered development environment, the development products, and lessons learned from interacting with the regulator. The deliverable will be the presentation and associated materials.

Task 5:

The Contractor shall develop a final report describing the work summarizing the work performed and the results and conclusions derived. The final report shall include findings and recommendations for future research.

C.5 Deliverables and Delivery Schedule

Deliverable	Due Date	Format	Submit to
Task 1 design	4 months after start of work	As appropriate to the tools used.	COR
Task 2 artifacts	6 months after start of work	As appropriate to the tools used.	COR
Task 3	The initial kickoff meeting for task 3 will be held within two weeks of delivery of the artifacts and should be scheduled ahead of time. The NRC will provide input within one month of the kickoff meeting. The Contractor response is to be provided within one month of receipt of NRC input. The NRC will provide input and hold the second meeting within one month of receipt of the Contractor responses.	As appropriate to the tools used.	COR
Task 4	The presentation is to be provided within 11 months of the start of work.	Slides (PowerPoint or Adobe Portable Document Format (PDF)) Training session	COR

Figure 5.4: NRC RFP Page 8

```
// title: Reactor Trip System high,assurance demonstrator.
// project: High Assurance Rigorous Digital Engineering for Nuclear Safety (HARDENS)
// copyright (C) 2021 Galois
// author: Joe Kiniry <kiniry@galois.com>

system Reactor Trip System (RTS)
The overall shape of the Reactor Trip System (RTS) is an archetypal
*sense,compute,actuate* architecture. Sensors are in the ‘Sensors‘
subsystem. They are read by the ‘Instrumentation‘ subsystem, which
contains four separate and independent ‘Instrumentation‘
components. The "Compute" part of the architecture is spread across
the ‘Actuation Logic‘ subsystem,which contains the two ‘Voting‘
components which perform the actuation logic itself,and the ‘Root‘
subsystem which contains the core computation and I/O components, and
the two separate and independent devices that drive actuators.

subsystem RTS Architecture (Architecture)
This RTS architecture specification includes all of the core
concepts inherent to NPP Instrumentation and Control systems.
A system architecture specification often includes a software,
hardware, network, and data architecture specifications.

subsystem RTS Hardware Artifacts (Hardware)
The physical hardware components that are a part of the HARDENS RTS
demonstrator.

subsystem RTS Implementation Artifacts (Implementation)
A summary of the tools, technologies, specifications, and implementations
relevant to this high,assurance demonstrator’s development and assurance.

subsystem RTS Requirements (Requirements)
All requirements that the RTS system must fulfill, as driven by the
IEEE 603,2018 standards and the NRC RFP.

subsystem RTS Properties (Properties)
All correctness and security properties of the RTS system are
specified in this subsystem.

subsystem IEEE Std 603,2018 Characteristics (Characteristics)
The IEEE 603,2018 requirements (known as "characteristics" in
the standard) which the RTS demonstrator system must fulfill.

relation RTS contains Architecture
relation RTS contains Hardware
relation RTS contains Properties
relation RTS contains Characteristics
```

Figure 5.5: A fragment of the RTS LANDO domain model

```
1  // Events are (seemingly,atomic, from the point of view of an external
2  // observer) interactions/state,transitions of the system. The full
3  // set of specified events characterizes every potential externally
4  // visible state change that the system can perform.
5
6
7  // External input actions are those that are triggered by external input on UI.
8  events Demonstrator External Input Actions
9
10 Manually Actuate Device
11 The user manually actuates a device.
12
13 Select Operating Mode
14 The user puts an instrumentation division in or takes a division out of 'maintenance' mode.
15
16 Perform Setpoint Adjustment
17 The user adjusts the setpoint for a particular channel in a particular division in
       ↪ maintenance mode.
18
19 Configure Bypass of an Instrument Channel
20 The user sets the mode of the channel of an instrumentation division to either bypass or
       ↪ normal mode.
21
22 Configure Active Trip Output State of an Instrument Channel
23 The user sets the mode of the channel of an instrumentation division to either trip or
       ↪ normal mode.
24
25 // External output actions are those that are triggered by internal
26 // state change, which is, in turn, sometimes prompted by external input
27 // actions.
28 events Demonstrator External Output Actions
```

Figure 5.6: A fragment of the RTS LANDO (external input) events

```
1  // Scenarios are sequences of events. Scenarios document normal and
2  // abnormal traces of system execution.
3
4  // Test scenarios are scenarios that validate a system conforms to its
5  // requirements through runtime verification (testing). Each scenario
6  // is refined to a (possibly parametrized) runtime verification
7  // property. If a testbench is complete, then every path of a
8  // system's state machine should be covered by the its set of scenarios.
9
10 scenarios Self,Test Scenarios
11
12 Normal Self,Test Behavior 1a ,Trip on Mock High Pressure Reading from that Pressure Sensor
13 The user selects 'maintenance' for an instrumentation division, the
14 division's pressure channel is set to 'normal' mode, the pressure
15 setpoint is set to a value v, the user simulates a pressure input to
16 that division exceeding v, the division generates a pressure trip.
17
18 Normal Self,Test Behavior 1b ,Trip on Environmental High Pressure Reading from that Pressure
       ↪ Sensor
19 The user selects 'maintenance' for an instrumentation division, the
20 division's pressure channel is set to 'normal' mode, the pressure
21 setpoint is set to a value v, the division reads a pressure sensor
22 value division exceeding v, the division generates a pressure trip.
23
24 Normal Self,Test Behavior 2a ,Trip on Mock High Temperature Reading from that Temperature
       ↪ Sensor
25 The user selects 'maintenance' for an instrumentation division, the
26 division's temperature channel is set to 'normal' mode, the
27 temperature setpoint is set to a value v, the user simulates a
28 temperature input to that division exceeding v, the division generates
29 a temperature trip.
```

Figure 5.7: A fragment of the RTS LANDO scenarios

```
-- We use three different C compilers for (cross-)compilation.
type compiler =
  { GCC, Clang, CompCert }

-- We target three different ISAs in software compilation because our
-- development platforms for the POSIX-based virtual platform is
-- either ARM or X86-based and the SoC digital twin and deployment
-- platform are RISC-V-based.
type isa =
  { ARM, X86, RV32 }

-- The feature model of the RTS demonstrator itself.

-- The cost of a demonstrator is expressed in U.S. dollars and is
-- based upon the value of the board plus all physical devices that
-- are attached. A purely virtualized RTS demonstrator has zero
-- hardware cost.

rts : kind of struct
  with -- Which development board is being used?
       board : dev_board
       -- How much does the hardware for this demonstrator cost in USD?
       cost : nat
       -- What level of assurance does the demonstrator have overall?
       assurance : assurance_level
       -- Is the FPGA being twinned via a Verilog simulator/emulator?
       soc : twin_or_physical
       -- Is the first tempurature sensor a twin or physically present?
       ts1 : twin_or_physical
       -- Is the second tempurature sensor a twin or physically present?
       ts2 : twin_or_physical
       -- Is the first pressure sensor a twin or physically present?
       ps1 : twin_or_physical
       -- Is the second pressure sensor a twin or physically present?
       ps2 : twin_or_physical
       -- Is the first actuator a twin or physically present?
       sa1 : twin_or_physical
       -- Is the second actuator a twin or physically present?
       sa2 : twin_or_physical
       -- Which C compiler is used to (cross-)compile the software?
       comp : compiler
       -- Which ISA is the compiler (cross-)compiling to?
       target : isa
       -- Are all devices twins?
       all_devices_twins : bool
       -- Should sensors be simulated?
       simulate_sensors : bool
```

Figure 5.8: A fragment of the RTS LOBOT feature model

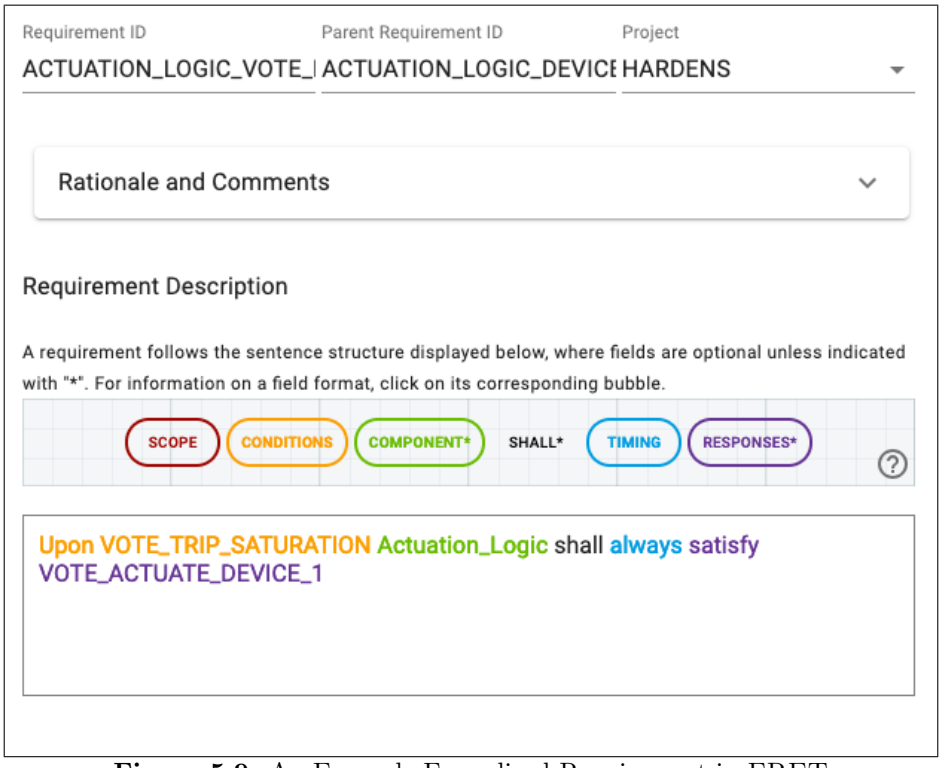

Figure 5.9: An Example Formalized Requirement in FRET

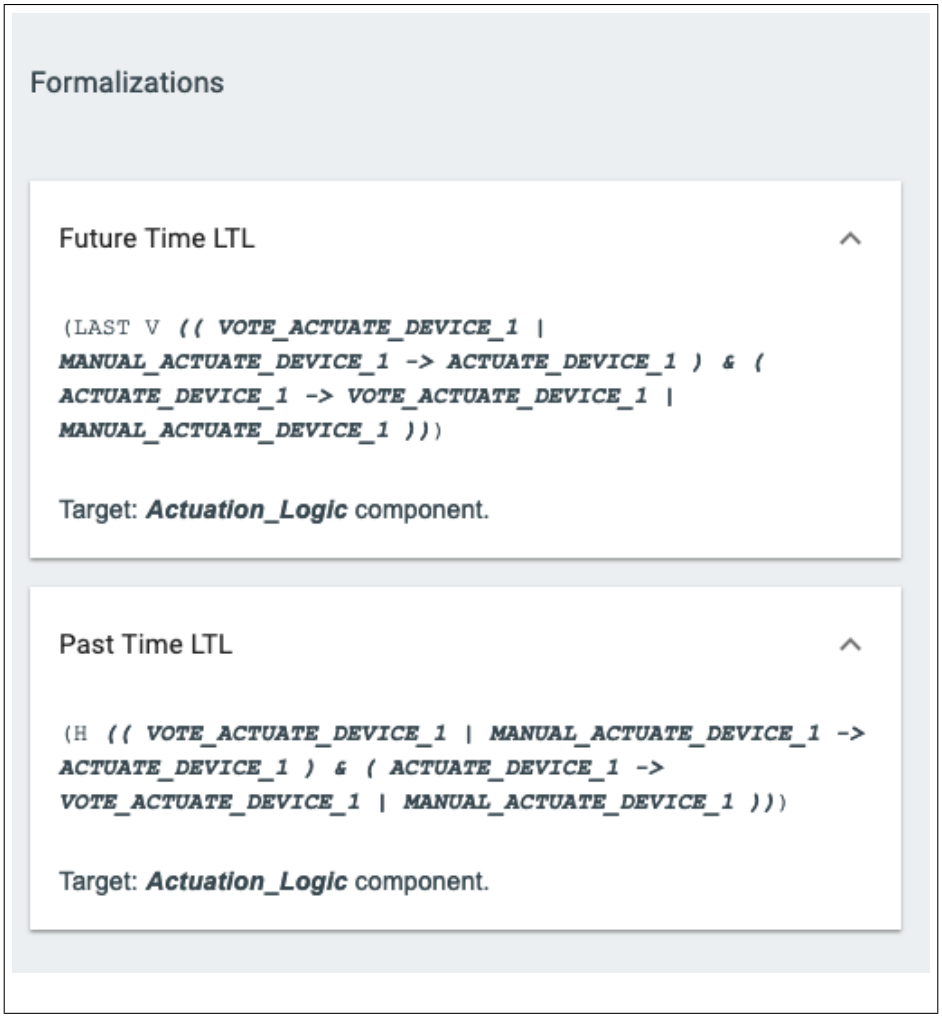

Figure 5.10: Some Example Requirement Semantics in FRET

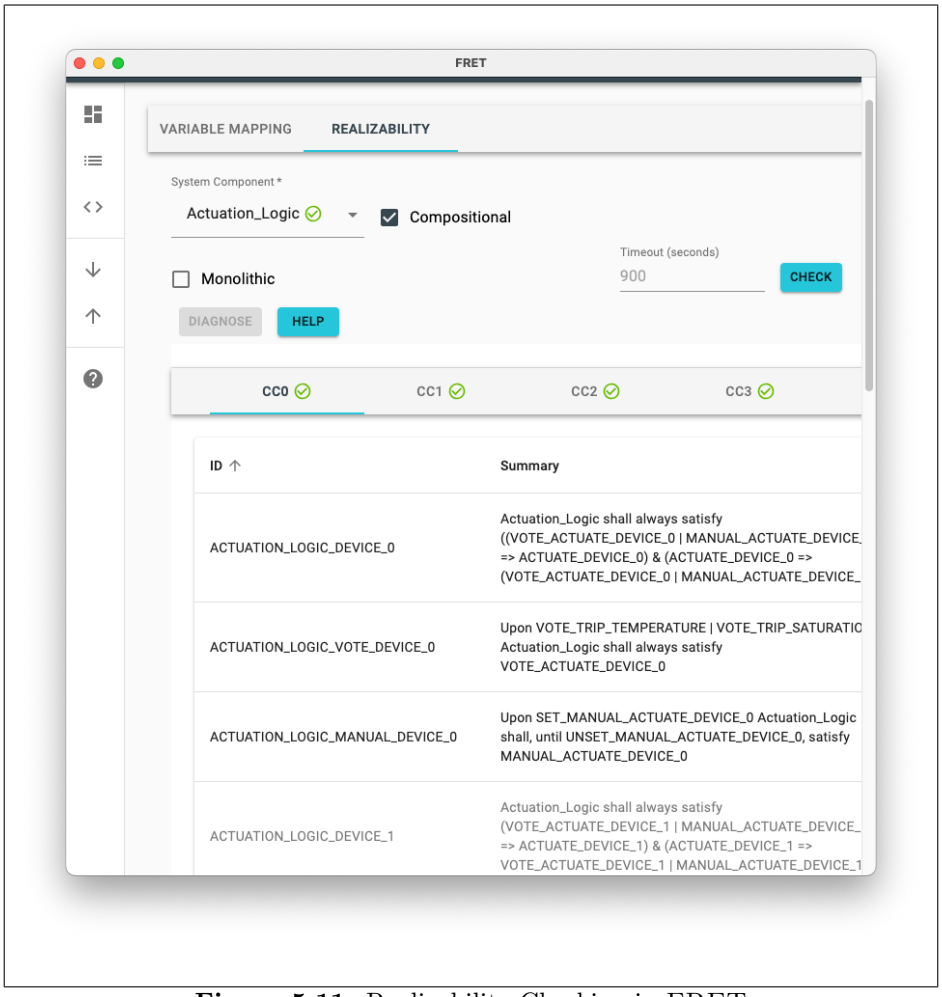

Figure 5.11: Realizability Checking in FRET

```
1  // HARDENS Reactor Trip System (RTS) Actuator Unit
2  // A formal model of RTS Actuator behavior written in the Cryptol
3  // DSL.
4  //
5  // @author Alex Bakst <abakst@galois.com>
6  // @created November, 2021
7  // @refines HARDENS.sysml
8  // @refines RTS.lando
9  // @refines RTS_Requirements.json
10
11 module RTS::Actuator where
12
13 type Actuation = Bit
14 type Mode = Bit
15
16 /** @requirements
17       ACTUATION_LOGIC_MANUAL_DEVICE_{0,1} satisfied by definition
18 */
19 type Actuator =
20   { input: Actuation
21   , manualActuatorInput: Actuation
22   }
23
24 SetInput: Actuation -> Actuator -> Actuator
25 SetInput on actuator = {actuator | input = on }
26
27 SetManual: Actuation -> Actuator -> Actuator
28 SetManual on actuator = {actuator | manualActuatorInput = on}
29
30 ActuateActuator : [2]Actuation -> Actuation
31 ActuateActuator inputs = (inputs @ (0:[1])) || (inputs @ (1:[1]))
```

Figure 5.12: CRYPTOL model of Actuator component

```
// HARDENS Reactor Trip System (RTS) Actuation Unit
// A formal model of RTS Actuation Unit behavior written in the
// Cryptol DSL.
//

module RTS::ActuationUnit where

import RTS::Utils
import RTS::InstrumentationUnit

type Input = [3][4]TripPort
type ActuationPort = Bit
type ActuationUnit = { output: [2]ActuationPort }
type CoincidenceLogic = [4]TripPort -> Bit
type OrLogic = [2] -> Bit

TemperatureLogic: [4]TripPort -> Bit
TemperatureLogic ts = Coincidence_2_4 ts

private
  Coincidence_2_4 : [4]TripPort -> Bit
  Coincidence_2_4 x =
    (a&&b) || ((a||b) && (c||d)) || (c&&d)
    where
      a = (x @ (0:[2])) != 0
      b = (x @ (1:[2])) != 0
      c = (x @ (2:[2])) != 0
      d = (x @ (3:[2])) != 0

  count : {n} (fin n, n >= 1) => [n] -> [width n]
  count bs = sum [ if b then 1 else 0 | b <- bs ]

  /** @requirements
        ACTUATION_LOGIC_VOTE_TEMPERATURE
  */
  property actuation_logic_vote_temperature (inp: Input) =
    (count [i != 0 | i <- (inp @ T)] >= 2) == TemperatureLogic (inp @ T)
```

Figure 5.13: CRYPTOL model of Actuation Unit component

```
NRC Understanding
Provide to the NRC expert technical services in order to develop a
better understanding of how Model,Based Systems Engineering (MBSE)
methods and tools can support regulatory reviews of adequate design
and design assurance.
```

Figure 5.14: An informal project goal: **NRC Understanding**

```
Demonstrator Parts
Our demonstrator includes high,assurance software and hardware,
includes open source RISC,V Central Processing Units.
```

Figure 5.15: A semi-formal project goal: **Demonstrator Parts**

```
1 // Both formal and rigorous consistency checks of the requirements
2 // will be accomplished by using false theorem checks and proofs in
3 // the Cryptol model and in software and hardware source code;
4 Requirements Consistency
5 Requirements must be shown to be consistent.
```

Figure 5.16: A formal project goal: **Requirements Consistency**

Chapter 6

Conclusion

This final report summarizes the outcomes of the *High Assurance Rigorous Digital Engineering for Nuclear Safety* (**HARDENS**) project, in which Galois has developed a high-assurance, safety-critical demonstration system for the Nuclear Regulatory Commission using Rigorous Digital Engineering (RDE). The system in question is a Digital Instrumentation and Control (DI&C) system for Nuclear Power Plants (NPPs), and is called the Reactor Trip System (RTS).

The report covered the following topics:

- In chapter 2, we covered the topic at a 50,000 ft level: what is this project about, what are the NRC's goals, just what are models and model-based engineering (MBE), what forces have influenced the evolution of models and MBE over the past few decades, how do assurance cases relate to MBE, and how is the system that was built to demonstrate modern MBE—the HARDENS demonstrator—described and assured in what follows.

- chapter 3 contains a top-level summary of model-based engineering in the 2020s, with particular attention on which concepts, tools, and technologies have been successfully adopted in high-end industry, especially for mission- and safety-critical systems development, assurance, and certification. This chapter also includes a gap analysis which details the challenges that remain for widespread adoption of MBE, or deep acceptance in a specific industrial segment, such as the critical infrastructure sector.

- chapter 4 discusses how one should review model-based systems, particularly from the point of view of carefully reviewing large, complex, high-assurance systems automatically and manually using both digital and paper-based means. These kinds of reviews are a natural part of the validation or certification regimes used by NIST and the NSA, as well as international certification regimes such as Common Criteria, but are not traditionally the remit of reviews that take place at the FAA, in the DoD branches (such as flight certification), or at the NRC.

- The HARDENS Reactor Trip System (RTS) Demonstrator is discussed in gory detail in chapter 5.
- After this chapter, a large appendix contains all of the cross-referenced models and code for the entire RTS demonstrator.

Were we to have used proprietary tools instead of, or in addition to, the open source tools that have been used in this project, then we would have chosen to use the following tools:

- Cameo Systems Modeler (CATIA) from Dassault Systèmes would have been used to create and reason about a SysML version 1 model, and generate hypertext digital and paper documentation from that model.
- OSATE2 from the Carnegie Mellon University's Software Engineering Institute (SEI) would have been used to create and reason about an AADL model.[1]
- A broad set of tools from Ansys, including ModelCenter, SCADE, Twin Builder, medina, and others, would have been used to (a) perform trade-study analysis and, (b) to implement and verify the instrumentation subsystem again using another language/technique for more heterogeneity.
- BigLever's onePLE would have been used to characterize our product line model and drive the aforementioned trade-study.
- And a tool like AdvoCate from NASA or ASCE from Adelard would have been used to document and review our assurance case.

6.1 What is Next for the RTS

This project is likely just the first of many that Galois hopes to do with the NRC. Its technical focus and outcomes are highly aligned with a rich area R&D at Galois that is seeing a lot of attention from the U.S. Government.

Technical Next Steps. On the technical side of things, the implementation of the RTS does not wholly conform to our original, ambitious, plans for demonstrating heterogeneous, independent high-assurance implementations.

While the RTS is fault tolerant and does fulfill all core RFP requirements and IEEE characteristics, the three core implementation (which was meant to demonstrate diversity and fault tolerance at the hardware compute level) of the RTS's System-on-Chip (SoC) is not complete. The single core version is complete and demonstrable.

We also intended to demonstrate multiple, heterogeneous techniques for interfacing with sensors and actuators (software and hardware device drivers),

[1] In fact, some of this work has already begun outside of the remit of this project with Dr. John Hatcliff and Dr. Robby at Kansas State University.

but those components, too, are not complete. Only single drivers have been experimented with in the FPGA-based demo.

The SoC has seen only lightweight exercising and runtime testing. No model-based testing or formal assurance of the SoC has been completed.

Multiple other projects and agencies have expressed an interest in reusing or extending this line of work. We hope to build on top of this foundation for the NRC, and we expect that, at least, DARPA will be using the RTS as a case study in teaching about, and R&D on, RDE.

Non-Technical Next Steps. On the non-technical side of things, there is work to do on matters related to the MBE market, MBE adoption, statues and rule-making around MBE, and more.

A part of the reason for the small market for high-assurance components or subsystems for NPPs are perceived or actual barriers to entry. If such barriers can be lowered by making licensing easier through clear guidance for modern digital I&C design techniques, such as those demonstrated in this project, then a marked market shift could occur.

Bibliography

[1] John M. Borky and Thomas H. Bradley. *Effective Model-Based Systems Engineering*. Springer, 2018.

[2] Peter H. Feiler and David P. Gluch. *Model-Based Engineering with AADL: An Introduction to the SAE Architecture Analysis & Design*. Addison-Wesley, 2012.

[3] Tim Weilkiens et al. *Model-Based System Architecture*. Wiley, 2022.

Appendix A

Lando Models

A.1 Top-level RTS Domain Engineering Model Structure

Listing A.1: Lando model of the top-level design.

```
1  // title: Reactor Trip System high,assurance demonstrator.
2  // project: High Assurance Rigorous Digital Engineering for Nuclear Safety (HARDENS)
3  // copyright (C) 2021 Galois
4  // author: Joe Kiniry <kiniry@galois.com>
5
6  system Reactor Trip System (RTS)(⊑ RTS (?? on page ??), RTS (?? on page ??), RTS
       (line 224 on page 156), RTS (line 19 on page 137))
7  The overall shape of the Reactor Trip System (RTS) is an archetypal
8  *sense,compute,actuate* architecture. Sensors are in the ‘Sensors‘
9  subsystem. They are read by the ‘Instrumentation‘ subsystem, which
10 contains four separate and independent ‘Instrumentation‘
11 components. The "Compute" part of the architecture is spread across
12 the ‘Actuation Logic‘ subsystem,which contains the two ‘Voting‘
13 components which perform the actuation logic itself,and the ‘Root‘
14 subsystem which contains the core computation and I/O components, and
15 the two separate and independent devices that drive actuators.
16
17 subsystem RTS Architecture (Architecture)(⊑ Architecture (line 26 on page 137),
       Architecture (line 7 on page 152), Architecture (line 136 on page 143))
18 This RTS architecture specification includes all of the core
19 concepts inherent to NPP Instrumentation and Control systems.
20 A system architecture specification often includes a software,
21 hardware, network, and data architecture specifications.
22
23 subsystem RTS Hardware Artifacts (Hardware)(⊑ Hardware (line 191 on page 144), RTS
       Hardware Artifacts (line 5 on page 146), Hardware (?? on page ??), Hardware
       (line 28 on page 137), RTS Hardware Artifacts (?? on page ??))
24 The physical hardware components that are a part of the HARDENS RTS
25 demonstrator.
26
27 subsystem RTS Implementation Artifacts (Implementation)(⊑ Artifacts (line 30 on
       page 137), Artifacts (line 1 on page 147))
28 A summary of the tools, technologies, specifications, and implementations
29 relevant to this high,assurance demonstrator's development and assurance.
30
31 subsystem RTS Requirements (Requirements)(⊑ Requirements (line 5 on page 149),
       Requirements (line 12 on page 149), Requirements (line 31 on page 137))
32 All requirements that the RTS system must fulfill, as driven by the
```

```
IEEE 603,2018 standards and the NRC RFP.

subsystem RTS Properties (Properties)(⊑ Properties (line 5 on page 148), Properties (line 32 on page 137))
All correctness and security properties of the RTS system are
specified in this subsystem.

subsystem IEEE Std 603,2018 Characteristics (Characteristics)(⊑ Characteristics (line 27 on page 149), Characteristics (line 34 on page 138), Characteristics (line 5 on page 139))
The IEEE 603,2018 requirements (known as "characteristics" in
the standard) which the RTS demonstrator system must fulfill.

relation RTS (line 6) contains Architecture (line 17)
relation RTS (line 6) contains Hardware (line 23)
relation RTS (line 6) contains Properties (line 35)
relation RTS (line 6) contains Characteristics (line 39)
```

A.2 Project Acronyms

Listing A.2: Lando model of the Acronyms.

```
subsystem Proposal Acronyms (Acronyms)
A list of words formed by combining the initial letters of a multipart name.

// Source: Frama,C website
component ISO ANSI C Specification Language (ACSL)
The ANSI/ISO C Specification Language (ACSL) is a behavioral specification language for C
    ↪ programs.

// Source: https://csrc.nist.gov/glossary/term/Application_Programming_Interface
component Application Programming Interface (API)
A system access point or library function that has a well,defined syntax and
is accessible from application programs or user code to provide well,defined functionality.

component Application,Specific Integrated Circuit (ASIC)(⊑ ASIC (line 216 on page 144))
Custom,designed and/or custom,manufactured integrated circuits.

component Commercial Off The Shelf (COTS)
Software and hardware that already exists and is available from commercial sources.

component Central Processing Unit (CPU)(⊑ CPU (line 15 on page 147), CPU (line 206 on page 144))
A CPU is the electronic circuitry that executes instructions comprising a computer program.

component Continuous Verification (CV)(⊑ CV (line 154 on page 143))

component Communicating Sequential Processes (CSP)

component Digital Engineering (DE)

component Digital Instrumentation \& Control (diandc)

component Defense Industrial Base (DIB)

component Department of Defense (DoD)

component Domain Specific Language (DSL)

component Electronic Design Automation (EDA)

component Field Programmable Gate Array (FPGA)(⊑ FPGA (?? on page ??), FPGA (line 228 on page 144))
```

```
39 |
40 | component Gnu Compiler Collection (GCC)
41 |
42 | component Government Furnished Equipment (GFE)
43 |
44 | component Galois Low,energy Asynchronous Secure SoC for Computer Vision (GLASS,CV)
45 |
46 | component General Purpose I/O (GPIO)(⊑ GPIO (line 218 on page 144))
47 |
48 | component High,Assurance Rigorous Digital Engineering for Nuclear Safety (HARDENS)
49 |
50 | component Hardware Description Language (HDL)(⊑ HDL (line 201 on page 144))
51 |
52 | component Hash,based Message Authentication Code (HMAC)
53 |
54 | component Higher,Order Logic (HOL)
55 |
56 | component Hardware Security Module (HSM)
57 |
58 | component Intelligence Community (IC)
59 |
60 | component Integrated Development Environment (IDE)
61 |
62 | component Intellectual Property (IP)
63 |
64 | component Instruction Set Architecture (ISA)
65 |
66 | component Intermediate Representation (IR)(⊑ IR (line 183 on page 144))
67 |
68 | component Java Modeling Language (JML)
69 |
70 | component Low Level Virtual Machine (LLVM)(⊑ LLVM (line 184 on page 144))
71 |
72 | component Model,Based Engineering (MBE)
73 |
74 | component Model,Based Systems Engineering (MBSE)
75 |
76 | component Natural Language Processing (NLP)
77 |
78 | component Nuclear Regulatory Commission (NRC)
79 |
80 | component National Security Agency (NSA)
81 |
82 | component Nuclear Power Plant (NPP)
83 |
84 | component Object Constraint Language (OCL)
85 |
86 | component Open Systems Architecture (OSA)
87 |
88 | component Power Performance Area and Security (PPAS)
89 |
90 | component Rigorous Digital Engineering (RDE)
91 |
92 | component Register Transfer Level (RTL)(⊑ RTL (line 28 on page 147))
93 |
94 | component Reactor Trip System (RTS)(⊑ RTS (?? on page ??), RTS (?? on page ??),
      RTS (line 224 on page 156), RTS (line 19 on page 137))
95 |
96 | component Software Analysis Workbench (SAW)
97 |
98 | component Safety Critical Application Development Environment (SCADE)
99 |
100 | component Secure Hash Algorithm (SHA)
101 |
102 | component Satisfiability Modulo Theories (SMT)
103 |
104 | component Secret Ninja Formal Methods (SNFM)
```

```
105
106 component Statement of Work (SoW)
107
108 component SystemVerilog (SV)(⊑ SystemVerilog (line 203 on page 144))
109
110 component SystemVerilog Assertions (SVA)
111
112 component Size Weight and Power (SWaP)(⊑ SWaP (line 192 on page 144))
113
114 component System Modeling Language (SysML)
115
116 component System,on,Chip (SoC)
117
118 component System Security Integration Through Hardware and Firmware (SSITH)
119
120 component User Interface (UI)
121
122 component Unified Modeling Language (UML)
123
124 component Universal Serial Bus (USB)(⊑ USB (line 225 on page 144))
125
126 component United States Government (USG)
127
128 component Unified Theories of Programming (UTP)
129
130 component Universal Verification Methodology (UVM)
131
132 component User eXperience (UX)
133
134 component Verilog(⊑ Verilog (line 204 on page 144))
135
136 component Vienna Development Method (VDM)
137
138 component Very High Speed Integrated Circuit (VHSIC)
139
140 component VHSIC Hardware Description Language (VHDL)
```

A.3 System Architecture

architecture.lando

Listing A.3: Lando model of the architecture.

```
1  // Architecture
2
3  subsystem RTS System Architecture (RTS_System_Arch)(⊑ RTS_System_Arch (line 19 on
       page 152))
4
5  subsystem Root(⊑ Root (?? on page ??))
6
7  component Core Finite State Machine (CFSM)(⊑ CFSM (?? on page ??))
8  inherit FSM
9
10 component Programming I/O (Programming_IO)(⊑ Programming_IO (?? on page ??))
11 inherit IO
12
13 component UI I/O (UI_IO)(⊑ UI_IO (?? on page ??))
14 inherit IO
15
16 component Debugging I/O (Debugging_IO)(⊑ Debugging_IO (?? on page ??))
17 inherit IO
18
19 subsystem Actuation Logic(⊑ Actuation_Logic (?? on page ??))
```

```
component Voting 1(⊑ Voting (line 254 on page 145))

component Voting 2(⊑ Voting (line 254 on page 145))

component Actuator 1(⊑ Actuator (line 196 on page 155), Actuator (line 222 on page 144), Actuator (?? on page ??))

component Actuator 2(⊑ Actuator (line 196 on page 155), Actuator (line 222 on page 144), Actuator (?? on page ??))

subsystem Computation(⊑ Computation (?? on page ??))

component RISC,V CPU 1(⊑ RISC-V CPU (?? on page ??))

component RISC,V CPU 2(⊑ RISC-V CPU (?? on page ??))

component RISC,V CPU 3(⊑ RISC-V CPU (?? on page ??))

subsystem Hardware(⊑ Hardware (?? on page ??), Hardware (line 191 on page 144), Hardware (line 28 on page 137))

subsystem FPGA(⊑ FPGA (?? on page ??), FPGA (line 228 on page 144))

component Lattice ECP,5 FGPA Development Board(⊑ DevBoard (line 26 on page 146))

subsystem Actuators(⊑ actuator2 (line 273 on page 156), Actuator (line 222 on page 144), actuator1 (line 272 on page 156), Actuator1 (?? on page ??), Actuator2 (?? on page ??), Actuator (line 196 on page 155), Actuator (?? on page ??), Actuators (?? on page ??))

component Actuator 1(⊑ Actuator (line 196 on page 155), Actuator (line 222 on page 144), Actuator (?? on page ??))

component Actuator 2(⊑ Actuator (line 196 on page 155), Actuator (line 222 on page 144), Actuator (?? on page ??))

subsystem Sensors(⊑ Sensor (line 219 on page 144), Sensors (?? on page ??), Sensor (line 21 on page 152))

component Temperature Sensor 1(⊑ Temperature Sensor (?? on page ??), Temperature Sensor (line 220 on page 144), Temperature Sensor (line 59 on page 153))

component Temperature Sensor 2(⊑ Temperature Sensor (?? on page ??), Temperature Sensor (line 220 on page 144), Temperature Sensor (line 59 on page 153))

component Pressure Sensor 1(⊑ Pressure Sensor (line 83 on page 153), Pressure Sensor (?? on page ??), Pressure Sensor (line 221 on page 144), PressureSensor (?? on page ??), PressureSensor (line 73 on page 153))

component Pressure Sensor 2(⊑ Pressure Sensor (line 83 on page 153), Pressure Sensor (?? on page ??), Pressure Sensor (line 221 on page 144), PressureSensor (?? on page ??), PressureSensor (line 73 on page 153))

subsystem Instrumentation(⊑ Instrumentation (?? on page ??), Instrumentation (?? on page ??), Instrumentation (line 100 on page 154), Instrumentation (?? on page ??))

component Instrumentation 1(⊑ Instrumentation (?? on page ??), Instrumentation (?? on page ??), Instrumentation (line 100 on page 154), Instrumentation (?? on page ??))

component Instrumentation 2(⊑ Instrumentation (?? on page ??), Instrumentation (?? on page ??), Instrumentation (line 100 on page 154), Instrumentation (?? on page ??))

component Instrumentation 3(⊑ Instrumentation (?? on page ??), Instrumentation (?? on page ??), Instrumentation (line 100 on page 154), Instrumentation (?? on page ??))

component Instrumentation 4(⊑ Instrumentation (?? on page ??), Instrumentation (??
```

```
        on page ??),   Instrumentation   (line 100 on page 154),   Instrumentation   (?? on
        page ??))
68  |
69  | // Top,level subsystems.
70  | relation RTS_System_Arch (line 3) contains Root (line 5)
71  | relation RTS_System_Arch (line 3) contains Actuation Logic (line 19)
72  | relation RTS_System_Arch (line 3) contains Computation (line 29)
73  | relation RTS_System_Arch (line 3) contains Hardware (line 37)
74  | relation RTS_System_Arch (line 3) contains Instrumentation (line 59)
75  |
76  | // Nested subsystems.
77  | relation Hardware (line 37) contains FPGA (line 39)
78  | relation Hardware (line 37) contains Actuators (line 43)
79  | relation Hardware (line 37) contains Sensors (line 49)
80  |
81  | // Client,supplier relations.
82  | relation Root (line 5) client Actuation Logic (line 19)
83  | relation Root (line 5) client Computation (line 29)
84  |
85  | relation Computation (line 29) client Hardware (line 37)
86  | relation Actuation Logic (line 19) client Hardware (line 37)
87  | relation Instrumentation (line 59) client Hardware (line 37)
88  | relation Instrumentation (line 59) client Actuation Logic (line 19)
89  | relation Actuation Logic (line 19) client Instrumentation (line 59)
```

A.4 System Dataflow

Listing A.4: Lando model of the dataflow.

```
1   | subsystem RTS Implementation Artifacts (Artifacts)(⊑  Artifacts   (line 30 on page 137),
        Artifacts   (line 1 on page 147))
2   |
3   | component Cryptol System Specification (CryptolSpec)(⊑  CryptolSpec   (line 257 on
        page 145))
4   | A specification of a model written in the Cryptol domain,specific
5   | language (DSL), either as Literate Cryptol, which can be Cryptol
6   | embedded in Markdown or LaTeX, or plain Cryptol. Cryptol is a strongly
7   | typed, functional DSL for specifying and reasoning about bit,level
8   | algorithms and their correctness properties and is mainly used to
9   | specify cryptographic algorithms. See https://crypto.net/ for more
10  | information.
11  |
12  | component Cryptol Software Compiler (CryptolToC)(⊑  CryptolToC   (line 7 on page 147))
13  | Multiple versions of a Cryptol software compiler exist which can
14  | compile different subsets of the Cryptol language into implementations
15  | and test benches written in the C, Java, and LLVM languages.
16  |
17  | component Cryptol Hardware Compiler (CryptolToSystemVerilog)(⊑  CryptolToSystemVerilog
        (line 11 on page 147))
18  | Multiple versions of a Cryptol hardware compiler exist which can
19  | compile different subsets of the Cryptol language into implementations
20  | and test benches written in the VHDL, Verilog, and SystemVerilog.
21  |
22  | component Software Implementation (Software)(⊑  Software   (line 267 on page 145))
23  |
24  | component Hand,written Software Implementation (SWImpl)(⊑  HWImpl   (line 274 on page 145),
        SWImpl   (line 269 on page 145))
25  | inherit Hand,written Software
26  |
27  | component Synthesized Software Implementation (SynthSW)(⊑  SynthHW   (line 275 on page 145),
        SynthSW   (line 271 on page 145))
28  | inherit Machine,generated Software
29  |
```

```
component Hardware Implementation (Hardware)(⊑ Hardware (line 191 on page 144), Hardware (line 28 on page 137), Hardware (?? on page ??), Hardware Implementation (line 273 on page 145))

component Hand,written Hardware Implementation (HWImpl)(⊑ HWImpl (line 274 on page 145), SWImpl (line 269 on page 145))
inherit Hand,written Hardware

component Synthesized Hardware Implementation (SynthHW)(⊑ SynthHW (line 275 on page 145), SynthSW (line 271 on page 145))
inherit Machine,generated Hardware

component COTS High,Assurance RV32I RISC,V CPU (CPU)(⊑ CPU (line 15 on page 147), CPU (line 206 on page 144))

component CompCert Compiler (CompCert)(⊑ CompCert (line 16 on page 147))

component Bluespec Compiler (BSC)(⊑ BSC (line 20 on page 147))

component SymbiFlow Synthesizer (SymbiFlow)(⊑ SymbiFlow (line 24 on page 147))

component Software Binaries (Binaries)(⊑ Binary (line 276 on page 145))

component Demonstrator Verilog (RTL)(⊑ RTL (line 28 on page 147))

component FPGA Bitstream (Bitstream)(⊑ Bitstream (line 284 on page 145))

subsystem Dataflow of RTS Implementation Artifacts (Dataflow)(⊑ Dataflow (line 30 on page 147))
This specification, which comes from the Galois HARDENS proposal,
describes the relationships between various levels of specifications,
implementations, and assurance artifacts for the HARDENS demonstrator.
indexing
  proposal_figure: 3
  figure_name: Dataflow of RTS Implementation Artifacts.

relation CryptolToC (line 12) client CryptolSpec (line 3)
relation CryptolToSystemVerilog (line 17) client CryptolSpec (line 3)

relation SynthSW (line 27) client CryptolToC (line 12)

relation SynthHW (line 35) client CryptolToSystemVerilog (line 17)
relation SynthHW (line 35) client BSC (line 42)

relation CompCert (line 40) client SynthSoftImpl
relation CompCert (line 40) client SoftImpl

relation BSC (line 42) inherit Compiler
relation BSC (line 42) client HWImpl (line 32)

relation SymbiFlow (line 44) client SynthHW (line 35)
relation SymbiFlow (line 44) client CPU (line 38)

relation Binaries (line 46) client CompCert (line 40)

relation RTL (line 48) client SymbiFlow (line 44)
relation RTL (line 48) contains Soft,core RISC,V CPU

relation Bitstream (line 50) contains SynthHW (line 35)
relation Bitstream (line 50) contains CPU (line 38)
relation Bitstream (line 50) client SymbiFlow (line 44)
```

A.5 System Events

Listing A.5: Lando model of the Events.

```
// Events are (seemingly,atomic, from the point of view of an external
// observer) interactions/state,transitions of the system. The full
// set of specified events characterizes every potential externally
// visible state change that the system can perform.

// External input actions are those that are triggered by external input on UI.
events Demonstrator External Input Actions

Manually Actuate Device
The user manually actuates a device.

Select Operating Mode
The user puts an instrumentation division in or takes a division out of 'maintenance' mode.

Perform Setpoint Adjustment
The user adjusts the setpoint for a particular channel in a particular division in
    ↪ maintenance mode.

Configure Bypass of an Instrument Channel
The user sets the mode of the channel of an instrumentation division to either bypass or
    ↪ normal mode.

Configure Active Trip Output State of an Instrument Channel
The user sets the mode of the channel of an instrumentation division to either trip or
    ↪ normal mode.

// External output actions are those that are triggered by internal
// state change, which is, in turn, sometimes prompted by external input
// actions.
events Demonstrator External Output Actions

Display Pressure(⊑ Display Pressure (line 62 on page 139))
The UI displays the current pressure reading for an instrumentation division.

Display Temperature(⊑ Display Temperature (line 63 on page 139))
The UI displays the current temperature reading for an instrumentation division.

Display Saturation Margin(⊑ Display Saturation Margin (line 64 on page 139))
The UI displays the current saturation margin reading for an instrumentation division.

Display Trip Output Signal State
The UI displays the current trip signal output for a particular channel and instrumentation
    ↪ division.

Display Indication of Channel in Bypass(⊑ Display Indication Of Channel in Bypass
    (line 66 on page 139))
The UI displays the current bypass mode for a particular channel and instrumentation
    ↪ division.

// Internal actions are those that are not triggered by external input on UI.
events Demonstrator Internal Actions

Trip on High Pressure
An instrumentation division reads a pressure sensor value that exceeds its setpoint and
    ↪ generates a trip output.

Trip on High Temperature
An instrumentation division reads a temperature sensor value that exceeds its setpoint and
    ↪ generates a trip output.

Trip on Low Saturation Margin
An instrumentation division reads temperature and pressure values such that the
saturation margin is below its setpoint and generates a trip output.

Vote on Like Trips using Two,out,of,four Coincidence(⊑ Vote (line 15 on page 138))
```

```
An actuation unit reads two like trip inputs and generates the corresponding automatic
    ↪ actuation signal.

Automatically Actuate Device(⊑ A (line 19 on page 138))
An actuation unit generates an automatic actuation signal and sends it to the corresponding
    ↪ device.

Self,test of Safety Signal Path(⊑ T (line 22 on page 138))
The RTS simulates inputs to a pair of instrumentation divisions and checks the corresponding
    ↪ actuation signals.
```

A.6 Project Glossary

Listing A.6: Lando model of the glossary.

```
subsystem Proposal Glossary (Glossary)(⊑ Glossary (line 10 on page 141))
A list of often difficult or specialized words with their definitions, often placed at the
    ↪ back of a book.

component Behavioral Interface Specification Language (BISL)(⊑ BISL (line 22 on
    page 141))
A formal, state,based specification language that focuses on the
specification of the interfaces of discrete modules in a system, and
often times includes model,based specification constructs to improve
usability and expressivity.

component BlueCheck(⊑ BlueCheck (line 17 on page 141))
A property,based testing framework for components written in Bluespec
SystemVerilog that uses some of the design patterns and ideas from the
QuickCheck.

component Coq(⊑ Coq (line 24 on page 141))
Coq is an interactive theorem prover first released in 1989. It allows
for expressing mathematical assertions, mechanically checks proofs of
these assertions, helps find formal proofs, and extracts a certified
program from the constructive proof of its formal specification. Coq
works within the theory of the calculus of inductive constructions, a
derivative of the calculus of constructions. Coq is not an automated
theorem prover but includes automatic theorem proving tactics
(procedures) and various decision procedures.

component Cryptol(⊑ Cryptol (line 25 on page 141))
Cryptol is a domain specific programming language for cryptography
developed by Galois. The language was originally developed for use by
the United States National Security Agency. The language is also used
by private firms that provide information technology systems, such as
Amazon and defense contractors in the United States. The programming
language is used for all aspects of developing and using cryptography,
such as the design and implementation of new ciphers and the
verification of existing cryptographic algorithms.

component DevSecOps(⊑ DevSecOps (line 26 on page 141))
The use of tools in a user's local and remote design, development,
validation, verification, maintenance, and evolution environments that
facilitate the automatic and continuous evaluation by static and
dynamic means of a system/subsystem/component's behavioral (e.g.,
safety and correctness) and non,behavioral (e.g., well,formedness and
security) properties.

component Digital Instrumentation and Control Systems (DIANDC)(⊑ DIANC (line 27 on
    page 141))
One of several types of control systems and associated instrumentation
used for industrial process control. Such systems can range in size
```

from a few modular panel,mounted controllers to large interconnected
and interactive distributed control systems with many thousands of
field connections. Systems receive data from remote sensors measuring
process variables (PVs), compare the collected data with desired
setpoints (SPs), and derive command functions which are used to
control a process through the final control elements (FCEs), such as
control valves.

component Formal Requirements Elicitation Tool (FRET)(⊑ FRET (line 31 on page 141), FRET (line 51 on page 150))
The NASA Formal Requirements Elicitation Tool is used to make writing,
understanding, and debugging formal requirements natural and
intuitive.

component PVS(⊑ PVS (line 41 on page 141))
A specification language integrated with support tools and an
automated theorem prover, developed at the Computer Science Laboratory
of SRI International. PVS is based on a kernel consisting of an
extension of Church's theory of types with dependent types, and is
fundamentally a classical typed higher,order logic.

component RISC,V
RISC,V (pronounced ``risk,five'') is an open standard instruction set
architecture (ISA) based on established reduced instruction set
computer (RISC) principles. Unlike most other ISA designs, the RISC,V
ISA is provided under open source licenses that do not require fees to
use. A number of companies are offering or have announced RISC,V
hardware, open source operating systems with RISC,V support are
available and the instruction set is supported in several popular
software toolchains.

component Requirements State Modeling Language (RSML)(⊑ RSML (line 53 on page 141))
A formal specification language that uses hierarchical finite state
machines to specify system requirements.

component SAT(⊑ SAT (line 58 on page 141))
The Boolean satisfiability problem (sometimes called propositional
satisfiability problem and abbreviated SAT) is the problem of
determining if there exists an interpretation that satisfies a given
Boolean formula.

component SAWscript(⊑ SAWscript (line 61 on page 141))
The proof script language is used to specify the assumptions and proof
goals of formal verifications to the SAW tool.

component soft,core
A CPU or SoC that is implemented in an HDL and synthesized to a
bitstream and loaded onto an FPGA.

component SPARK(⊑ SPARK (line 74 on page 142))
A formally defined computer programming language based on the Ada
programming language, intended for the development of high integrity
software used in systems where predictable and highly reliable
operation is essential. It facilitates the development of applications
that demand safety, security, or business integrity.

component SpeAR(⊑ SpeAR (line 77 on page 142))
An integrated development environment for formally specifying and
rigorously analyzing requirements.

component Verifier for Concurrent C (VCC)(⊑ VCC (line 84 on page 142))
VCC is a program verification tool that proves correctness of
annotated concurrent C programs or finds problems in them. VCC extends
C with design by contract features, like pre, and postcondition as
well as type invariants. Annotated programs are translated to logical
formulas using the Boogie tool, which passes them to an automated SMT
solver Z3 to check their validity.

```
112 |
113 | component Verified Software Toolchain (VST)(⊑ VST (line 92 on page 142))
114 | A software toolchain that includes static analyzers to check
115 | assertions about a C program; optimizing compilers to translate a C
116 | program to machine language; and operating systems and libraries to
117 | supply context for the C program. The Verified Software Toolchain
118 | project assures with machine,checked proofs that the assertions
119 | claimed at the top of the toolchain really hold in the
120 | machine,language program, running in the operating,system context.
121 |
122 | component Refinement(⊑ Refinement (line 95 on page 142))
123 |
124 | component Property(⊑ Property (line 96 on page 142))
125 |
126 | component Safety Property(⊑ Safety Property (line 97 on page 142))
127 | inherit Property
128 |
129 | component Correctness Property(⊑ Correctness Property (line 98 on page 142))
130 | inherit Property
131 |
132 | component Security Property(⊑ Security Property (line 99 on page 142))
133 | inherit Property
134 |
135 | component Model(⊑ Model (line 100 on page 142))
136 |
137 | component Semi,Formal Model(⊑ Semi-Formal Model (line 101 on page 142))
138 | inherit Model
139 |
140 | component Formal Model(⊑ Formal Model (line 102 on page 142))
141 | inherit Model
142 |
143 | component Consistent(⊑ Consistent (line 103 on page 142))
144 | inherit Property
145 |
146 | component Complete(⊑ Complete (line 104 on page 142))
147 | inherit Property
148 |
149 | component Consistent Model(⊑ Consistent Model (line 105 on page 142))
150 | inherit Model Consistent
151 |
152 | component Complete Model(⊑ Complete Model (line 106 on page 142))
153 | inherit Model Complete
154 |
155 | component Requirement(⊑ Requirement (line 121 on page 142))
156 |
157 | component Scenario(⊑ Scenario (line 122 on page 142))
158 |
159 | component Product(⊑ Product (line 123 on page 142))
160 |
161 | component Product Line(⊑ Product Line (line 124 on page 142))
162 |
163 | component Configure(⊑ Configure (line 125 on page 142))
164 |
165 | component DOORS(⊑ DOORS (line 126 on page 142))
166 |
167 | component Clafer(⊑ Clafer (line 127 on page 142))
168 |
169 | component Lobot(⊑ Lobot (line 128 on page 142))
170 |
171 | component Lando
172 |
173 | component Denotational(⊑ Denotational (line 108 on page 142))
174 |
175 | component Operational(⊑ Operational (line 109 on page 142))
176 |
177 | component Semantics(⊑ Semantics (line 110 on page 142))
```

```
178 |
179 | component Risk(⊑ Risk (line 146 on page 143))
180 |
181 | component Power(⊑ Power (line 147 on page 143))
182 |
183 | component Resource(⊑ Resource (line 148 on page 143))
184 |
185 | component Reliability(⊑ Reliability (line 149 on page 143))
186 |
187 | component Rigorous(⊑ Rigorous (line 114 on page 142))
188 | A specification that has a precise, unambiguous, formal semantics
189 | grounded in real world formal foundations and systems engineering
190 | artifacts, such as source code and hardware designs.
191 |
192 | component Collaborative Development Environment (CDE)(⊑ CDE (line 152 on page 143))
193 |
194 | component Continuous Integration (CI)(⊑ CI (line 153 on page 143))
195 |
196 | component Continuous Verification (CV)(⊑ CV (line 154 on page 143))
197 |
198 | component Analyzer(⊑ Analyzer (line 155 on page 143))
199 |
200 | component Static Analyzer(⊑ Static Analyzer (line 156 on page 143))
201 | inherit Analyzer
202 |
203 | component Dynamic Analyzer(⊑ Dynamic Analyzer (line 157 on page 143))
204 | inherit Analyzer
205 |
206 | component Finite State Machine (FSM)(⊑ FSM (line 129 on page 143))
207 |
208 | component Deterministic(⊑ Deterministic (line 115 on page 142))
209 |
210 | component Non,deterministic(⊑ Non-deterministic (line 116 on page 142))
211 |
212 | component Deterministic Finite State Machine (DFSM)(⊑ DFSM (line 130 on page 143))
213 | inherit FSM Deterministic
214 |
215 | component Non,deterministic Finite State Machine (NFSM)(⊑ NFSM (line 132 on page 143))
216 | inherit FSM Non,deterministic
217 |
218 | component Abstract State Machine (ASM)(⊑ ASM (line 134 on page 143))
219 | inherit FSM
220 |
221 | component Design(⊑ Design (line 135 on page 143))
222 |
223 | component Architecture(⊑ Architecture (line 26 on page 137), Architecture (line 7 on
      page 152), Architecture (line 136 on page 143))
224 |
225 | component Specification(⊑ Specification (line 137 on page 143))
226 |
227 | component Architecture Specification(⊑ Architecture Specification (line 138 on page 143))
228 | inherit Specification
229 |
230 | component Solver(⊑ Solver (line 158 on page 143))
231 |
232 | component Formal Method (FM)(⊑ FM (line 117 on page 142))
233 |
234 | component Logical Framework (LF)(⊑ LF (line 159 on page 143))
235 |
236 | component Programming Language (PL)(⊑ PL (line 26 on page 151), PL (line 178 on
      page 143))
237 |
238 | component Specification Language(⊑ Specification Language (line 164 on page 143))
239 |
240 | component Protocol(⊑ Protocol (line 165 on page 143))
241 |
242 | component System Specification(⊑ System Specification (line 166 on page 143))
```

```
inherit Specification

component Hand,written(⊑ Hand-written (line 167 on page 143))

component Machine,generated(⊑ Machine-generated (line 168 on page 143))

component Source,level Specification Language(⊑ Source-level Specification Language
    (line 169 on page 143))
inherit Specification Language

component Model,based Specification Language(⊑ Model-based Specification Language
    (line 171 on page 143))
inherit Specification Language

component System(⊑ System (line 139 on page 143))

component Distributed System(⊑ Distributed System (line 140 on page 143))
inherit System

component Concurrent System(⊑ Concurrent System (line 141 on page 143))
inherit System

component Cryptographic Protocol(⊑ Cryptographic Protocol (line 174 on page 143))
inherit Protocol

component Cryptographic Algorithm(⊑ Cryptographic Algorithm (line 175 on page 143))

component I/O (IO)(⊑ IO (line 217 on page 144))

component General Purpose I/O (GPIO)(⊑ GPIO (line 218 on page 144))
inherit IO

component Sensor(⊑ Sensor (line 219 on page 144), Sensors (?? on page ??), Sensor
    (line 21 on page 152))

component Actuator(⊑ actuator2 (line 273 on page 156), Actuator (line 222 on
    page 144), actuator1 (line 272 on page 156), Actuator1 (?? on page ??),
    Actuator2 (?? on page ??), Actuator (line 196 on page 155), Actuator (?? on
    page ??), Actuators (?? on page ??))

component Solenoid(⊑ Solenoid (line 223 on page 144))
inherit Actuator

component Compiler(⊑ Compiler (line 185 on page 144))

component Synthesizer(⊑ Synthesizer (line 200 on page 144))

component Universal Serial Bus (USB)(⊑ USB (line 225 on page 144))

component LED(⊑ LED (line 226 on page 144))

component Cable(⊑ Cable (line 227 on page 144))

component Program(⊑ Program (line 143 on page 143))

component Bitstream(⊑ Bitstream (line 284 on page 145))

component Field,Programmable Gate Array (FPGA)(⊑ FPGA (?? on page ??), FPGA
    (line 228 on page 144))

component ECP,5(⊑ ECP-5 (line 229 on page 144))
inherit FPGA

component Printed Component Board (PCB)(⊑ PCB (line 230 on page 144))

component Connector(⊑ Connector (line 214 on page 144))

component USB Connector(⊑ USB Connector (line 233 on page 144))
```

```
304
305 component USB Mini Connector (USB,Mini)(⊑ USB_Mini (line 234 on page 144))
306 inherit USB Connector
307
308 component High,Assurance(⊑ High-Assurance (line 160 on page 143))
309
310 component C(⊑ C (line 180 on page 143))
311
312 component PMOD(⊑ PMOD (line 235 on page 144))
313
314 component JTAG(⊑ JTAG (line 236 on page 144))
315
316 component Driver(⊑ Driver (line 237 on page 144))
317
318 component Voting(⊑ Voting (line 254 on page 145))
319
320 component UCB Cable(⊑ USB Cable (line 245 on page 145))
321 inherit USB Cable
322
323 component Output LED(⊑ Output LED (line 16 on page 146))
324 inherit LED
```

A.7 System Hardware

Listing A.7: Lando model of the system hardware.

```
1  subsystem RTS Hardware Artifacts(⊑ RTS Hardware Artifacts (line 5 on page 146),
       Hardware (line 28 on page 137), RTS Hardware Artifacts (?? on page ??))
2  The physical hardware components that are a part of the HARDENS RTS
3  demonstrator.
4
5  component USB Cable(⊑ USB Cable (line 245 on page 145))
6  A normal USB cable.
7  What kind of USB connector is on the start of the cable?
8  What kind of USB connector is on the end of the cable?
9
10 relation USB Cable (line 5) inherit USB, Cable
11
12 component SERDES Test SMA Connectors (J9,J26)(⊑ SERDES Test SMA Connector (line 10 on
       page 146))
13
14 component Parallel Config Header (J38)(⊑ Parallel Config Header (line 11 on page 146),
       J38 (line 28 on page 146))
15
16 component Versa Expansion Connectors (J39,J40)(⊑ Versa Expansion Connector (line 12 on
       page 146))
17
18 component SPI Flag Configuration Memory (U4)(⊑ SPI Flag Configuration Memory (line 13 on
       page 146), U4 (line 30 on page 146))
19
20 component CFG Switches (SW1)(⊑ SW1 (line 31 on page 146))
21
22 component Input Switches (SW5)(⊑ Input Switch (line 15 on page 146), SW5 (line 32 on
       page 146))
23
24 component Output LEDs (D5,D12)(⊑ Output LED (line 16 on page 146))
25
26 component Input Push Buttons (SW2,SW4)(⊑ Input Push Button (line 17 on page 146))
27
28 component 12 V DC Power Input (J37)
29
30 component GPIO Headers (J32,J33)(⊑ GPIO Headers (line 19 on page 146))
31
```

```
component PMOD/GPIO Header (J31)(⊑ PMOD/GPIO Header (line 20 on page 146), J31 (line 37 on page 146))

component Microphone Board/GPIO Header (J30)(⊑ Microphone Board/GPIO Header (line 21 on page 146), J30 (line 38 on page 146))

component Prototype Area(⊑ Prototype Area (line 39 on page 146))

component ECP5,5G Device (U3)(⊑ ECP5-5G Device (line 22 on page 146), U3 (line 40 on page 146))

component JTAG Interface (J1)(⊑ JTAG Interface (line 24 on page 146), J1 (line 41 on page 146))

component Mini USB Programming (J2)(⊑ Mini USB Programming (line 25 on page 146), J2 (line 42 on page 146))

component Lattice ECP,5 FPGA Development Board (Board)(⊑ DevBoard (line 26 on page 146))

component Temperature Sensor(⊑ Temperature Sensor (?? on page ??), Temperature Sensor (line 220 on page 144), Temperature Sensor (line 59 on page 153))
A sensor that is capable of measuring the temperature of its environment.
What is your temperature reading in Celsius (C)?

component Pressure Sensor(⊑ Pressure Sensor (line 83 on page 153), Pressure Sensor (?? on page ??), Pressure Sensor (line 221 on page 144), PressureSensor (?? on page ??), PressureSensor (line 73 on page 153))
A sensor that is capable of measuring the air pressure of its environment.
What is your pressure reading in Pascal (P)?

component Solenoid Actuator(⊑ Solenoid Actuator (line 50 on page 146))
A solenoid actuator capable of being in an open or closed state.
Close!
Open!

relation Temperature Sensor (line 46) inherit Sensor
relation Pressure Sensor (line 50) inherit Sensor
relation Solenoid Actuator (line 54) inherit Actuator

subsystem Physical Architecture(⊑ Physical Architecture (line 2 on page 147))
The physical architecture of the HARDENS RTS demonstrator.

component USB UI Cable (UI,C)
The USB cable used to communicate the ASCII UI to/from the board.

component USB Programming Cable (Prog,C)
The USB cable used to program the board with a bitstream.

component USB Debugging I/O Cable (Debug,C)
The USB cable used to interact with the board in a debugger.

component Dev Board (Board)
A PCB developer board used to prototype hardware.

// * MOSFET power control kit: https://www.sparkfun.com/products/12959

// * 12 V Latch solenoid: https://www.sparkfun.com/products/15324

// * Pressure sensor: https://www.sparkfun.com/products/11084

component Temperature Sensor 1 (TS1)(⊑ Temperature Sensor (?? on page ??), Temperature Sensor (line 220 on page 144), Temperature Sensor (line 59 on page 153))
The first of two redundant temperature sensors.

component Temperature Sensor 2 (TS2)(⊑ Temperature Sensor (?? on page ??), Temperature Sensor (line 220 on page 144), Temperature Sensor (line 59 on page 153))
The second of two redundant temperature sensors.
```

```
component Pressure Sensor 1 (PS1)(⊑ Pressure Sensor (line 83 on page 153), Pressure Sensor (?? on page ??), Pressure Sensor (line 221 on page 144), PressureSensor (?? on page ??), PressureSensor (line 73 on page 153))
The first of two redundatnt pressure sensors.

component Pressure Sensor 2 (PS2)(⊑ Pressure Sensor (line 83 on page 153), Pressure Sensor (?? on page ??), Pressure Sensor (line 221 on page 144), PressureSensor (?? on page ??), PressureSensor (line 73 on page 153))
The second of two redundant pressure sensors.

component Solenoid Actuator 1 (SA1)(⊑ Solenoid Actuator (line 50 on page 146))
The first of two redundant solenoid actuators.

component Solenoid Actuator 2 (SA2)(⊑ Solenoid Actuator (line 50 on page 146))
The second of two redundant solenoid actuators.

component HARDENS Demonstrator (Demonstrator)
The fully assembled HARDENS demonstrator hardware with all component
present.

component Developer Machine
The computer used by a developer to interface with the demonstrator,
typically for driving the demonstrator's UI and programming and
debugging the board.

relation Demonstrator (line 102) client Board (line 75)
relation Board (line 75) client UI
relation Board (line 75) client UI-C (line 66)
relation Board (line 75) client Prog-C (line 69)
relation Board (line 75) client Debug-C (line 72)
relation Board (line 75) client TS1 (line 84)
relation Board (line 75) client TS2 (line 87)
relation Board (line 75) client PS1 (line 90)
relation Board (line 75) client PS2 (line 93)
relation Board (line 75) client SA1 (line 96)
relation Board (line 75) client SA2 (line 99)
relation UI-C (line 66) client Developer Machine (line 106)
relation Prog-C (line 69) client Developer Machine (line 106)
relation Debug-C (line 72) client Developer Machine (line 106)
relation U3 (line 38) inherit FPGA
relation Board (line 75) contains ECP5,5G
relation Board (line 75) inherit PCB
relation FPGA Dev Board contains J2 (line 42)
```

A.8 System Instrumentation

Listing A.8: Lando model of the system instrumentation.

```
subsystem RTS Instrumentation Architecture
The architecture for the instrumentation (sensors and actuators)
subsystem of the RTS demonstrator.

subsystem RTS Instrumentation Systems Architecture
The systems architecture for the instrumentation subsystem of the RTS
demonstrator. Some of the architecture is implemented in hardware,
and some is implementated in software.

component Instrumentation Implementation (InstImpl) inherit Driver
A software or hardware driver that interfaces with a sensor. In the
RTS demonstrator there are two kinds of sensors: pressure and
temperature.

component Actuator Implementation (ActImpl)
```

```
16 inherit Driver
17 A software or hardware driver that interfaces with an actuator. In
18 the RTS demonstrator there is one kind of actuator: a solenoid.
19
20 component Voting Implementation (VoteImpl)
21 inherit Voting
22 A software or hardware implemenation of our voting algorithm that
23 provides fault tolerance for decision,making based upon the attached
24 components' inputs.
25
26 subsystem Instrumentation Software Stack (SWStack)
27 inherit Software
28 The software stack associated with the instrumentation subsystem.
29
30 component Instrumentation Implementation 1 (InstImpl1) inherit Instrumentation
       ↪ Implementation
31 The first of four sensor drivers for the instrumentation subsystem.
32
33 relation InstImpl1 inherit SWImpl
34 relation InstImpl1 inherit High,Assurance
35 relation InstImpl1 inherit C
36
37 component Instrumentation Implementation 2 (InstImpl2)
38 inherit InstImpl
39 The second of four sensor drivers for the instrumentation subsystem.
40 There are multiple sensors in the architecture to provide fault
41 tolerance.
42
43 component Actuator Implementation 1 (ActImpl1)
44 inherit ActImpl
45 The first of two actuator drivers for the instrumentation subsystem.
46 There are multiple actuators in the architecture to provide fault
47 tolerance.
48
49 component Voting Implementation 1 (VoteImpl1)
50 inherit VoteImpl
51 The first of two implementations of the voting component. Voting is
52 used to implement redundancy of instrumentation and control in the RTS
53 demonstrator.
54
55 relation SWStack (line 26) client Binaries
56 relation Binaries client SWStack (line 26)
57
58 subsystem Instrumentation Actuation and Voting Hardware Stack (HWStack)
59 The hardware implementations driving a redundant subset of sensors,
60 actuators, and voting components.
61
62 component Instrumentation Implementation 3 (InstImpl3)
63 inherit InstImpl
64 The third of four sensor drivers for the instrumentation subsystem.
65 There are multiple sensors in the architecture to provide fault
66 tolerance.
67
68 component Instrumentation Implementation 4 (InstImpl4)
69 inherit InstImpl
70 The fourth of four sensor drivers for the instrumentation subsystem.
71 There are multiple sensors in the architecture to provide fault
72 tolerance.
73
74 component Actuator Implementation 2 (ActImpl2)
75 inherit ActImpl
76 The second of two actuator drivers for the instrumentation subsystem.
77 There are multiple actuators in the architecture to provide fault
78 tolerance.
79
80 component Voting Implementation 2 (VoteImpl2)
```

```
inherit VoteImpl
The second of two implementations of the voting component. Voting is
used to implement redundancy of instrumentation and control in the RTS
demonstrator.

relation HWStack (line 58) client Bitstream
relation Bitstream client HWStack (line 58)
```

A.9 Project Requirements

Listing A.9: Lando model of the project requirements.

```
// All requirements that the RTS system must fulfill, as driven by the
// IEEE 603,2018 standards and the NRC RFP.

requirements HARDENS Project High,level Requirements
// The high,level requirements for the project stipulated by the NRC RFP.

NRC Understanding(⊑ NRC Understanding (line 19 on page 149))
Provide to the NRC expert technical services in order to develop a
better understanding of how Model,Based Systems Engineering (MBSE)
methods and tools can support regulatory reviews of adequate design
and design assurance.

Identify Regulatory Gaps(⊑ Identify Regulatory Gaps (line 20 on page 149))
Identify any barriers or gaps associated with MBSE in a regulatory
review of Digital Instrumentation and Control Systems for existing
Nuclear Power Plants.

Demonstrate(⊑ Demonstrate (line 21 on page 149))
Galois will demonstrate to the Nuclear Regulatory Commission (NRC)
cutting,edge capabilities in the model,based design, validation, and
verification of safety,critical, mission,critical, high,assurance
systems.

Demonstrator Parts(⊑ Demonstrator Parts (line 22 on page 149))
Our demonstrator includes high,assurance software and hardware,
includes open source RISC,V Central Processing Units.

Demonstrator Groundwork(⊑ Demonstrator Groundwork (line 23 on page 149))
Our demonstrator lays the groundwork for a high,assurance reusable
product for safety critical Digital Instrumentation and Control
Systems systems in Nuclear Power Plants.

requirements NRC Characteristics
// The requirements driven by the IEEE 603,2018 standard for NPP I&C
// systems.

// Both formal and rigorous consistency checks of the requirements
// will be accomplished by using false theorem checks and proofs in
// the Cryptol model and in software and hardware source code;
Requirements Consistency(⊑ Requirements Consistency (line 6 on page 139), Requirements
    Consistency (line 35 on page 149))
Requirements must be shown to be consistent.

// A rigorous completeness validation of the requirements will be
// accomplished by demonstrating traceability from the project
// specification (including the RFP text describing the reactor trip
// system) to the formal models of the system and its properties.
Requirements Colloquial Completeness(⊑ Requirements Colloquial Completeness (line 36 on
    page 149), Requirements Colloquial Completeness (line 9 on page 139))
The system must be shown to fulfill all requirements.

```

```
// A formal verification of completeness of the requirements will be
// accomplished by using the chosen requirements checking tool
Requirements Formal Completeness(⊑ Requirements Formal Completeness (line 37 on
    page 149), Requirements Formal Completeness (line 12 on page 139))
Requirements must be shown to be formally complete.

// This characteristic will be demonstrated architecturally via the
// decoupling of computation across the two RISC,V instrumentation
// cores and two instrumentation units running on the FPGA.
Instrumentation Independence(⊑ Instrumentation Independence (line 15 on page 139),
    Instrumentation Independence (line 38 on page 149))
Independence among the four divisions of instrumentation (inability
for the behavior of one division to interfere or adversely affect the
performance of another).

// This characteristic will be demonstrated architecturally by
// decoupling the compute and I/O channels of the units from one
// another.
Channel Independence(⊑ Channel Independence (line 39 on page 149), Channel Independence
    (line 20 on page 139))
Independence among the two instrumentation channels within a division
(inability for the behavior of one channel to interfere or adversely
affect the performance of another).

// This characteristic will be demonstrated architecturally by
// partitioning the actuation logic across software and hardware
// units.
Actuation Independence(⊑ Actuation Independence (line 25 on page 139), Actuation
    Independence (line 40 on page 149))
Independence among the two trains of actuation logic (inability for
the behavior of one train to interfere or adversely affect the
performance of another).

// This characteristic will be demonstrated by rigorous validation via
// runtime verification and formal verification of the model and its
// implementation, as discussed in detail below.
Actuation Correctness(⊑ Actuation Correctness (line 30 on page 139), Actuation
    Correctness (line 41 on page 149))
Completion of actuation whenever coincidence logic is satisfied or
manual actuation is initiated.

// This characteristic will be demonstrated architecturally by
// partitioning the actuation logic across software and hardware
// units.
Self,Test/Trip Independence(⊑ Self-Test/Trip Independence (line 34 on page 140),
    Self-Test/Trip Independence (line 42 on page 149))
Independence between periodic self,test functions and trip functions
(inability for the behavior of the self,testing to interfere or
adversely affect the trip functions).
```

A.10 System Requirements

Listing A.10: Lando model of the requirements.

```
subsystem Requirements
The requirements for the RTS system are specified in the following
requirements specifications:
   HARDENS Project High,level Requirements
   NRC Characteristics.

// At the moment all system requirements are written in FRET. See the
// top,level [](../README.md) for more information.

// During Task 2 we will reify those same requirements into our Lando
// and SysML specifications as well.
```

A.11 System Behavioral Scenarios

Listing A.11: Lando model of the scenarios.

```
// Scenarios are sequences of events. Scenarios document normal and
// abnormal traces of system execution.

// Normal scenarios describe the normal behavior of the system when
// it is not in maintenance or self,test mode.

scenarios RTS Scenarios

Normal Behavior 1 ,Stable Normal State
The system is in the normal operating mode across all instrumentation
units, no unit is in test mode, and no unit is in maintenance mode, a
threshold setpoint v_p has been set for pressure, and a threshold
setpoint v_t has been set for temperature.

Normal Behavior 2 ,Event Control
The system is in a stable normal state and responds to a new event.

Normal Behavior 3 ,Sense Compute Actuate
The system is in a stable normal state and receives input from the
user and reads data from its sensors and, reacts to user input, and if
necessary, actuates an actuator.

Normal Behavior 4 ,Test Instrumentation
Test system behavior across all possible combinations of modes,
commands, pressure values, temperature values, and failure conditions.

Normal Behavior 5 ,Test Voting
Test system voting behavior across all possible combinations of vote
inputs.
```

A.12 System Test Scenarios

Listing A.12: Lando model of the Test scenarios.

```
// Scenarios are sequences of events. Scenarios document normal and
// abnormal traces of system execution.

// Test scenarios are scenarios that validate a system conforms to its
// requirements through runtime verification (testing). Each scenario
// is refined to a (possibly parametrized) runtime verification
// property. If a testbench is complete, then every path of a
// system's state machine should be covered by the its set of scenarios.

scenarios Self,Test Scenarios

Normal Self,Test Behavior 1a ,Trip on Mock High Pressure Reading from that Pressure Sensor
The user selects 'maintenance' for an instrumentation division, the
division's pressure channel is set to 'normal' mode, the pressure
setpoint is set to a value v, the user simulates a pressure input to
that division exceeding v, the division generates a pressure trip.

Normal Self,Test Behavior 1b ,Trip on Environmental High Pressure Reading from that Pressure
    ↪ Sensor
The user selects 'maintenance' for an instrumentation division, the
division's pressure channel is set to 'normal' mode, the pressure
setpoint is set to a value v, the division reads a pressure sensor
value division exceeding v, the division generates a pressure trip.

Normal Self,Test Behavior 2a ,Trip on Mock High Temperature Reading from that Temperature
    ↪ Sensor
The user selects 'maintenance' for an instrumentation division, the
division's temperature channel is set to 'normal' mode, the
temperature setpoint is set to a value v, the user simulates a
temperature input to that division exceeding v, the division generates
a temperature trip.

Normal Self,Test Behavior 2a ,Trip on Environmental High Temperature Reading from that
    ↪ Temperature Sensor
The user selects 'maintenance' for an instrumentation division, the
division's temperature channel is set to 'normal' mode, the
temperature setpoint is set to a value v, the division reads a
temperature sensor value division exceeding v, the division generates
a temperature trip.

Normal Self,Test Behavior 3a ,Trip on Mock Low Saturation Margin
The user selects 'maintenance' for an instrumentation division, the
division's saturation channel is set to 'normal' mode, the saturation
margin setpoint is set to a value v, the user simulates pressure and
temperature inputs to that division such that the saturation margin is
below v, the division generates a saturation margin trip.

Normal Self,Test Behavior 3a ,Trip on Environmental Low Saturation Margin
The user selects 'maintenance' for an instrumentation division, the
division's saturation channel is set to 'normal' mode, the saturation
margin setpoint is set to a value v, the division reads pressure and
temperature sensor values in that division such that the saturation
margin is below v, the division generates a saturation margin trip.

Normal Self,Test Behavior 4 ,Vote on Every Possible Like Trip
The user selects 'maintenance' for each instrumentation division, the
user configures a subset of channels to place in active trip output.

Normal Self,Test Behavior 5a ,Automatically Actuate All Mock Devices in Sequence
The user selects 'maintenance' for each instrumentation division, the
user places enough channels in 'active trip' to actuate a mock device.

Normal Self,Test Behavior 5b ,Automatically Actuate All Hardware Devices in Sequence
The user selects 'maintenance' for each instrumentation division, the
user places enough channels in 'active trip' to actuate a hardware
```

```
device.

Normal Self,Test Behavior 6 ,Manually Actuate Each Device in Sequence
The user manually actuates each device.

Normal Self,Test Behavior 7a ,Select Maintenance Operating Mode for each Division
The user selects 'maintenance' mode for each instrumentation division
in sequence.

Normal Self,Test Behavior 7b ,Select Normal Operating Mode for each Division
The user exits 'maintenance' mode for each instrumentation division in
sequence.

Normal Self,Test Behavior 8 ,Perform Each Kind of Setpoint Adjustment
For each instrumentation division, the user provides a setpoint for
temperature, then the user provides a setpoint for pressure, then the
user provides a setpoint for saturation margin.

Normal Self,Test Behavior 9 ,Configure Bypass of Each Instrument Channel in Sequence
For each instrumentation division, the user puts each channel in
'normal' and then 'bypass' mode.

Normal Self,Test Behavior 10 ,Configure Active Trip Output State of Each Instrument Channel
    ↪ in Sequence
For each instrumentation division, the user puts each channel in
'trip' and then 'normal' mode.

Normal Self,Test Behavior 11 ,Display Pressure Temperature and Saturation Margin
The user provides inputs for an instrumentation division, then the UI
displays the updated pressure, temperature and saturation margin.

Normal Self,Test Behavior 12 ,Display Every Trip Output Signal State in Sequence
The user configures an instrumentation division in 'maintenance',
selects a channel and places it in 'trip', the UI displays the trip
state, the user places the channel in 'normal' mode, the UI displays
the trip state.

Normal Self,Test Behavior 13 ,Display Indication of Every Channel in Bypass in Sequence
The user configures an instrumentation division in 'maintenance', the
user selects a channel and places it in bypass, the UI displays the
bypass state, the user places the channel in normal mode, the UI
displays the updated state.

Normal Self,Test Behavior 14 ,Demonstrate Periodic Continual Self,test of Safety Signal Path
The system starts, eventually the UI displays that self test has run.

Normal Behavior Full Self,Test
The system selects two instrumentation divisions and a device, the system
simulates inputs to the selected instrumentation divisions, the system checks
that the correct actuation signal would be sent to the selected device.

// Exceptional behaviors are *predictable* system behaviors triggered
// by external environment affects. Examples include a CPU, OS,
// device, or network total or partial failure. E.g., CPU deadlock,
// out,of,memory errors, permanent store partial and total failures,
// lack of space in permanent store, partial and total network
// failures, device driver partial or total failures, etc.

// In our architecture, every part of the RTS system, and particularly
// every part of the instrumentation and sensing subsystem and every
// part of the actuation subsystem must be able to fail and the system
// must respond appropriately.

// This is not detectable
Exceptional Behavior 1a ,Cause Actuator 1 to Fail
Manually actuate Actuator 1, the actuator fails, an error is indicated.
```

```
128
129 // This is not detectable
130 Exceptional Behavior 1b ,Cause Actuator 2 to Fail
131 Manually actuate Actuator 2, the actuator fails, an error is indicated.
132
133 // This is not detectable
134 Exceptional Behavior 1c ,Non,determinisitically Cause an Actuator to Eventually Fail
135 Repeatedly manually actuate/unactuate an Actuator, the actuator fails, an error is indicated
      ↪ .
136
137 Exceptional Behavior 2a ,Cause Temperature Sensor 1 to Fail
138 Instrumentation units 1 and 2 reports temperature readings that are inconsistent with
      ↪ Instrumentation units 3 and 4, the
139 discrepancy is indicated on the UI.
140
141 Exceptional Behavior 2b ,Cause Temperature Sensor 2 to Fail
142 Instrumentation units 1 and 2 reports temperature readings that are inconsistent with
      ↪ Instrumentation units 3 and 4, the
143 discrepancy is indicated on the UI.
144
145 Exceptional Behavior 2c ,Non,deterministically Cause a Temperature Sensor to Eventually Fail
146 Instrumentation units 1 and 2 reports temperature readings that are inconsistent with
      ↪ Instrumentation units 3 and 4, the
147 discrepancy is indicated on the UI.
148
149 Exceptional Behavior 3a ,Cause Pressure Sensor 1 to Fail
150 Instrumentation units 1 and 2 reports pressure readings that are inconsistent with
      ↪ Instrumentation units 3 and 4, the
151 discrepancy is indicated on the UI.
152
153 Exceptional Behavior 3b ,Cause Pressure Sensor 2 to Fail
154 Instrumentation units 1 and 2 reports pressure readings that are inconsistent with
      ↪ Instrumentation units 3 and 4, the
155 discrepancy is indicated on the UI.
156
157 Exceptional Behavior 3c ,Non,deterministically Cause a Pressure Sensor to Eventually Fail
158 Temperature Sensor 1 or Temperature Sensor 2 report inconsistent values,
159 discrepancy is indicated on the UI.
160
161 // Can be further refined, e.g.:
162 // Sensor values are read by Instrummetation Unit 1, the unit generates the wrong trip signal
      ↪ ,
163 Exceptional Behavior 4a ,Cause Instrumentation Unit 1 to Fail
164 Instrumentation unit 1 fails, self test indicates that a test has failed.
165
166 // Can be further refined, e.g.:
167 // Sensor values are read by Instrummetation Unit 1, the unit generates the wrong trip signal
      ↪ ,
168 Exceptional Behavior 4b ,Cause Instrumentation Unit 2 to Fail
169 Instrumentation unit 2 fails, self test indicates that a test has failed.
170
171 // Can be further refined, e.g.:
172 // Sensor values are read by Instrummetation Unit 1, the unit generates the wrong trip signal
      ↪ ,
173 Exceptional Behavior 4c ,Cause Instrumentation Unit 3 to Fail
174 Instrumentation unit 3 fails, self test indicates that a test has failed.
175
176 // Can be further refined, e.g.:
177 // Sensor values are read by Instrummetation Unit 1, the unit generates the wrong trip signal
      ↪ ,
178 Exceptional Behavior 4d ,Cause Instrumentation Unit 4 to Fail
179 Instrumentation unit 4 fails, self test indicates that a test has failed.
180
181 Exceptional Behavior 4e ,Non,Deterministically Cause Instrumentation Unit to Eventually Fail
182 At least one instrumentation unit fails to generate the appropriate trip signal, self test
      ↪ runs,
183 self test indicates that a test has failed.
```

```
184
185 Exceptional Behavior 5a ,Cause Temperature Demultiplexor 1 to Fail
186 At least one of Instrumentation units 1 or 2 report temperature readings inconsistent with
      ↪ instrumentation units 3 and 4.
187
188 Exceptional Behavior 5b ,Cause Temperature Demultiplexor 2 to Fail
189 At least one of Instrumentation units 3 or 4 report temperature readings inconsistent with
      ↪ instrumentation units 1 and 2.
190
191 Exceptional Behavior 5c ,Cause Pressure Demultiplexor 1 to Fail
192 At least one of Instrumentation units 1 and 2 report pressure readings inconsistent with
      ↪ instrumentation units 3 and 4.
193
194 Exceptional Behavior 5d ,Cause Pressure Demultiplexor 2 to Fail
195 At least one of Instrumentation units 3 and 4 report pressure readings inconsistent with
      ↪ instrumentation units 1 and 2.
196
197 Exceptional Behavior 5e ,Cause a Temperature Demultiplexor to Eventually Fail
198 All instrumentation units report consistent temperature readings, a demultiplexor fails, at
      ↪ least one of the
199 instrumentation units reports an inconsistent temperature reading.
200
201 Exceptional Behavior 5f ,Cause a Pressure Demultiplexor to Eventually Fail
202 All instrumentation units report consistent temperature readings, a demultiplexor fails, at
      ↪ least one of the
203 instrumentation units reports an inconsistent pressure reading.
```

A.13 System Tool Scenarios

Listing A.13: Lando model of the System Tool scenarios.

```
1  scenarios Tool Scenarios
2
3  Verify Software
4  Formally verify that software or firmware programs fulfill their
5  specifications.
6
7  Verify Hardware
8  Formally verify that a hardware design fulfills its specifications.
9
10 Formal Equivalence Checking
11 Formally verify that programs written in different languages (even
12 across the hardware/software boundary) are equivalent.
13
14 Symbolic Testing
15 Improve the assurance of software using symbolic testing.
16
17 Backend Solver Libraries
18 Provide libraries for symbolic formula representation and solver
19 interaction.
20
21 Binary Analysis
22 Analyze binaries in a variety of formats and for a host of different
23 Instruction Set Architectures.
24
25 Binary Rewriting
26 Perform binary analysis and rewriting for a variety of purposes.
27
28 Model,Based Test Generation
29 Automatically generate model,based tests for a software, firmware, or
30 hardware system.
31
```

```
Specify Semi,Formal Architecture using Natural Language Processing
Specify a systems architectures at a high level leveraging Natural
Language Processing technology.

Concretize Model
Create or generate a new model or an implementation from a semi,formal or formal
model or implementation by adding extra information, typically turning
a denotational property into an operational computation, and guarantee
that the new, refined model behaves identically to the previous,
abstract model.

Abstract Model
Extract formal models,,,including behavioral and architectural
models,,,from source code and binaries, and guarantee that all
properties of the abstract model hold for the more concrete model or
implementation.

Define Refinement Relation
Define a pair of functions, an abstraction function L: I,>M and
concretization function C: M,>I, such that they form a refinement
relation over some property P (roughly, P(c(l(i)))=P(i)) between their
pair of types M and I.

Formally Refine a Semi,formal Architecture
Specify and formally refine a semi,formal architecture.

Product Line Engineering
Specify and reason about product lines of hardware, firmware, and/or
software systems.

Reason about Products
Reason about products derived from product lines, particularly
automatically generated CPUs and SoCs.

Reason about Non,Behavioral Properties
Reason about non,behavioral properties of models or implementations,
such as security proofs of cryptographic algorithms and protocols,
safety and progress properties of concurrent or distributed systems,
information leakage properties of embedded systems and hardware
designs.

Configure Product Line
Make feature selections in a feature model in order to specify the
subset of products from a product line that are of interest.

Fully Configure Product Line
Configure a feature model until no open choices exist, thereby
creating a fully configured feature model that specifies a single
product.
```

Appendix B

Lobot Model

Listing B.1: Lobot-Model of RTS.

```
-- title: Reactor Trip System high-assurance demonstrator.
-- project: High Assurance Rigorous Digital Engineering for Nuclear Safety (HARDENS)
-- copyright (C) 2021 Galois
-- author: Joe Kiniry <kiniry@galois.com>

nat : kind of int where self >= 0

-- Our development platforms for running the RTS demonstrator in a
-- fully virtualized (Twin) mode. If we choose to target a real RV32,
-- then we will be running on the bare metal.

type virtualized_platform_runtime =
  { Posix, RV32_bare_metal, None }

-- The developer boards we have to choose from. We are using the
-- ECP-5 5G 85F variant of the Lattice Semiconductor dev board, and if
-- we choose to put the demonstrator on a real RV32, we will likely
-- use the Vega board.

type dev_board =
  { Virtual, LFE5UM5G_85F_EVN, RV32M1_VEGA, None }

-- The ECP-5 FPGA comes in several flavors. We are using the 5G
-- variant for this project. Other variants should be able to use the
-- exact same build chain.

type fpga =
  { ECP5, ECP5_5G }

-- We can assign an assurance level of every sub-component of the
-- system. This definition is made here to provide an enumeration of
-- assurance levels which we will assign/update later as assurance
-- work goes on.

type assurance_level =
  { None, Low, Medium, High }

-- Every subsystem and the system overall is realized either by a
-- physical component (e.g., a real sensor, actuator, or FPGA) or a
-- "Digital Twin", which is a simulation/emulation of the component in
-- question.

```

```
type twin_or_physical =
  { Twin, Physical }

-- Twins come in different fidelity levels.

-- "Perfect" fidelity means that our simulator/emulation is capable of
-- executing the actual software/hardware of the system, subsystem, or
-- component in such detail that all requirements can be validated or
-- verified in the twin as accurately as in the physical realization.

-- "High" fidelity means that we are executing the actual
-- software/hardware in question in a simulator or emulator that
-- replicates most, but not all, of the underlying functionality and
-- behavior of the device in question. For example, a cycle-accurate
-- Verilog simulator is high-fidelity, but is not "Perfect" fidelity
-- if we are concerned about EM side-channels.

-- A "Medium" fidelity twin also executes the actual
-- software/hardware, but elides non-behavioral properties that are
-- critical to fulfilling all system requirements. A hardware virtual
-- platform (VP) or an event-based Verilog simulator or emulator are
-- two examples of medium-fidelity digital twin environments.

-- A "Low" fidelity twin is an executable model that is usually fully
-- decoupled from the implementation. In order for the model to be
-- refinement-consistent with regards to more concrete models or the
-- software/hardware implementation, all measurable properties of the
-- model which relate to system requirements must hold through the
-- refinement.

type twin_fidelity =
  { Low, Medium, High, Perfect }

-- We use three different C compilers for (cross-)compilation.
type compiler =
  { GCC, Clang, CompCert }

-- We target three different ISAs in software compilation because our
-- development platforms for the POSIX-based virtual platform is
-- either ARM or X86-based and the SoC digital twin and deployment
-- platform are RISC-V-based.
type isa =
  { ARM, X86, RV32 }

-- The feature model of the RTS demonstrator itself.

-- The cost of a demonstrator is expressed in U.S. dollars and is
-- based upon the value of the board plus all physical devices that
-- are attached. A purely virtualized RTS demonstrator has zero
-- hardware cost.

rts : kind of struct
  with -- Which development board is being used?
       board : dev_board
       -- How much does the hardware for this demonstrator cost in USD?
       cost : nat
       -- What level of assurance does the demonstrator have overall?
       assurance : assurance_level
       -- Is the FPGA being twinned via a Verilog simulator/emulator?
       soc : twin_or_physical
       -- Is the first tempurature sensor a twin or physically present?
       ts1 : twin_or_physical
       -- Is the second tempurature sensor a twin or physically present?
       ts2 : twin_or_physical
       -- Is the first pressure sensor a twin or physically present?
       ps1 : twin_or_physical
```

```
109       -- Is the second pressure sensor a twin or physically present?
110       ps2 : twin_or_physical
111       -- Is the first actuator a twin or physically present?
112       sa1 : twin_or_physical
113       -- Is the second actuator a twin or physically present?
114       sa2 : twin_or_physical
115       -- Which C compiler is used to (cross-)compile the software?
116       comp : compiler
117       -- Which ISA is the compiler (cross-)compiling to?
118       target : isa
119       -- Are all devices twins?
120       all_devices_twins : bool
121       -- Should sensors be simulated?
122       simulate_sensors : bool
123       -- Should we use parallel execution?
124       parallel_execution : bool
125       -- Compile with automatic self-test mode enabled?
126       self_test_enabled : bool
127       -- Compile with debugging options?
128       debug : bool
129       -- Is this instance of the RTS fully virtualized and running only in software?
130       virtualized_platform_rt : bool
131       -- What development platform is being used to run this fully virtualized twin?
132       rt : virtualized_platform_runtime
133     where
134       cost = 0 | cost = 100 | cost = 200
135       all_devices_twins <=> ((ts1 = Twin) & (ts2 = Twin) & (ps1 = Twin) & (ps2 = Twin) & (sa1 =
              ↪  Twin) & (sa2 = Twin))
136       (soc = Twin) => all_devices_twins
137       virtualized_platform_rt <=> ((soc = Twin) & (board = RV32M1_VEGA) & (rt = None)) ^ ((soc
              ↪ = Twin) & (board = None))
138
139   -- @todo kiniry Add appropriate constraints for the comp and isa features.
140
141   virtualized_rts_configs : kind of rts
142     where all_devices_twins = true & cost = 0 & board = None & virtualized_platform_rt = true
143
144   -- Virtualized builds do not need a development board.
145   check_twin_build_configs : check
146     on c : virtualized_rts_configs
147     that c.board = None
```

Appendix C

FRET Specification

Listing C.1: RTS requirements from FRET.

```
[
  {
    "reqid" :"ACTUATION_ACTUATOR_0",
    "parent_reqid" :"",
    "rationale" :"RFP function 5,6\n",
    "fulltext" :"RTS shall always satisfy\n (Auto_Actuate_0_Actuator_0 | Auto_Actuate_1
        ↪ _Actuator_0 | UI_Manual_Actuate_Actuator_0)\n = Actuate_Actuator_0\n",
    "description" :"ENFORCED: in the interval defined by the entire execution.\nTRIGGER:
        ↪ first point in the interval.\nREQUIRES: for every trigger, RES must hold at all
        ↪ time points between (and including) the trigger and the end of the interval."
  },
  {
    "reqid" :"ACTUATION_ACTUATOR_1",
    "parent_reqid" :"",
    "rationale" :"RFP function 5,6\n",
    "fulltext" :"RTS shall always satisfy\n (Auto_Actuate_0_Actuator_1 | Auto_Actuate_1
        ↪ _Actuator_1 | UI_Manual_Actuate_Actuator_1)\n = Actuate_Actuator_1\n",
    "description" :"ENFORCED: in the interval defined by the entire execution.\nTRIGGER:
        ↪ first point in the interval.\nREQUIRES: for every trigger, RES must hold at all
        ↪ time points between (and including) the trigger and the end of the interval."
  },
  {
    "reqid" :"ACTUATION_AUTO_LOGIC_0_ACTUATOR_0",
    "parent_reqid" :"ACTUATION_ACTUATOR_0",
    "rationale" :"RFP function 4,5,RFP actuation logic architecture\n",
    "fulltext" :"RTS shall always satisfy if (Coincidence_0_T | Coincidence_0_P) then
        ↪ Auto_Actuate_0_Actuator_0\n",
    "description" :"ENFORCED: in the interval defined by the entire execution.\nTRIGGER:
        ↪ first point in the interval.\nREQUIRES: for every trigger, RES must hold at all
        ↪ time points between (and including) the trigger and the end of the interval."
  },
  {
    "reqid" :"ACTUATION_AUTO_LOGIC_0_ACTUATOR_1",
    "parent_reqid" :"ACTUATION_ACTUATOR_1",
    "rationale" :"RFP function 4,5,RFP actuation logic architecture\n",
    "fulltext" :"RTS shall always satisfy if Coincidence_0_S then Auto_Actuate_0_Actuator_1",
    "description" :"ENFORCED: in the interval defined by the entire execution.\nTRIGGER:
        ↪ first point in the interval.\nREQUIRES: for every trigger, RES must hold at all
        ↪ time points between (and including) the trigger and the end of the interval."
  },
  {
    "reqid" :"COINCIDENCE_LOGIC_0_T",
    "parent_reqid" :"ACTUATION_AUTO_LOGIC_0_ACTUATOR_0",
    "rationale" :"RFP function 4",
```

```
34      "fulltext" :"RTS shall always satisfy\n ((Trip_T_0 & Trip_T_1)\n | ((Trip_T_0 | Trip_T_1)
            ↪  & (Trip_T_2 | Trip_T_3))\n | (Trip_T_2 & Trip_T_3))\n = Coincidence_0_T\n",
35      "description" :"ENFORCED: in the interval defined by the entire execution.\nTRIGGER:
            ↪ first point in the interval.\nREQUIRES: for every trigger, RES must hold at all
            ↪ time points between (and including) the trigger and the end of the interval."
36    },
37    {
38      "reqid" :"COINCIDENCE_LOGIC_0_P",
39      "parent_reqid" :"ACTUATION_AUTO_LOGIC_0_ACTUATOR_0",
40      "rationale" :"RFP function 4",
41      "fulltext" :"RTS shall always satisfy\n ((Trip_P_0 & Trip_P_1)\n | ((Trip_P_0 | Trip_P_1)
            ↪  & (Trip_P_2 | Trip_P_3))\n | (Trip_P_2 & Trip_P_3))\n = Coincidence_0_P\n",
42      "description" :"ENFORCED: in the interval defined by the entire execution.\nTRIGGER:
            ↪ first point in the interval.\nREQUIRES: for every trigger, RES must hold at all
            ↪ time points between (and including) the trigger and the end of the interval."
43    },
44    {
45      "reqid" :"COINCIDENCE_LOGIC_0_S",
46      "parent_reqid" :"ACTUATION_AUTO_LOGIC_0_ACTUATOR_1",
47      "rationale" :"RFP function 4",
48      "fulltext" :"RTS shall always satisfy\n ((Trip_S_0 & Trip_S_1)\n | ((Trip_S_0 | Trip_S_1)
            ↪  & (Trip_S_2 | Trip_S_3))\n | (Trip_S_2 & Trip_S_3))\n = Coincidence_0_S\n",
49      "description" :"ENFORCED: in the interval defined by the entire execution.\nTRIGGER:
            ↪ first point in the interval.\nREQUIRES: for every trigger, RES must hold at all
            ↪ time points between (and including) the trigger and the end of the interval."
50    },
51    {
52      "reqid" :"ACTUATION_AUTO_LOGIC_1_ACTUATOR_0",
53      "parent_reqid" :"ACTUATION_ACTUATOR_0",
54      "rationale" :"RFP function 4,5,RFP actuation logic architecture\n",
55      "fulltext" :"RTS shall always satisfy if (Coincidence_1_T | Coincidence_1_P) then
            ↪ Auto_Actuate_1_Actuator_0\n",
56      "description" :"ENFORCED: in the interval defined by the entire execution.\nTRIGGER:
            ↪ first point in the interval.\nREQUIRES: for every trigger, RES must hold at all
            ↪ time points between (and including) the trigger and the end of the interval."
57    },
58    {
59      "reqid" :"ACTUATION_AUTO_LOGIC_1_ACTUATOR_1",
60      "parent_reqid" :"ACTUATION_ACTUATOR_1",
61      "rationale" :"RFP function 4,5,RFP actuation logic architecture\n",
62      "fulltext" :"RTS shall always satisfy if Coincidence_1_S then Auto_Actuate_1_Actuator_1",
63      "description" :"ENFORCED: in the interval defined by the entire execution.\nTRIGGER:
            ↪ first point in the interval.\nREQUIRES: for every trigger, RES must hold at all
            ↪ time points between (and including) the trigger and the end of the interval."
64    },
65    {
66      "reqid" :"COINCIDENCE_LOGIC_1_T",
67      "parent_reqid" :"ACTUATION_AUTO_LOGIC_1_ACTUATOR_0",
68      "rationale" :"RFP function 4",
69      "fulltext" :"RTS shall always satisfy\n ((Trip_T_0 & Trip_T_1)\n | ((Trip_T_0 | Trip_T_1)
            ↪  & (Trip_T_2 | Trip_T_3))\n | (Trip_T_2 & Trip_T_3))\n = Coincidence_1_T\n",
70      "description" :"ENFORCED: in the interval defined by the entire execution.\nTRIGGER:
            ↪ first point in the interval.\nREQUIRES: for every trigger, RES must hold at all
            ↪ time points between (and including) the trigger and the end of the interval."
71    },
72    {
73      "reqid" :"COINCIDENCE_LOGIC_1_P",
74      "parent_reqid" :"ACTUATION_AUTO_LOGIC_1_ACTUATOR_0",
75      "rationale" :"RFP function 4",
76      "fulltext" :"RTS shall always satisfy\n ((Trip_P_0 & Trip_P_1)\n | ((Trip_P_0 | Trip_P_1)
            ↪  & (Trip_P_2 | Trip_P_3))\n | (Trip_P_2 & Trip_P_3))\n = Coincidence_1_P\n",
77      "description" :"ENFORCED: in the interval defined by the entire execution.\nTRIGGER:
            ↪ first point in the interval.\nREQUIRES: for every trigger, RES must hold at all
            ↪ time points between (and including) the trigger and the end of the interval."
78    },
79    {
80      "reqid" :"COINCIDENCE_LOGIC_1_S",
81      "parent_reqid" :"ACTUATION_AUTO_LOGIC_1_ACTUATOR_1",
```

```
    "rationale" :"RFP function 4",
    "fulltext" :"RTS shall always satisfy\n ((Trip_S_0 & Trip_S_1)\n | ((Trip_S_0 | Trip_S_1)
        ↪  & (Trip_S_2 | Trip_S_3))\n | (Trip_S_2 & Trip_S_3))\n = Coincidence_1_S\n",
    "description" :"ENFORCED: in the interval defined by the entire execution.\nTRIGGER:
        ↪ first point in the interval.\nREQUIRES: for every trigger, RES must hold at all
        ↪ time points between (and including) the trigger and the end of the interval."
  },
  {
    "reqid" :"INSTRUMENTATION_0_T_TRIP",
    "parent_reqid" :"",
    "rationale" :"RFP function 1,2,3,9\n",
    "fulltext" :"When\n (!Bypass_T_0 & (T_0 > Setpoint_T_0))\nRTS shall always satisfy
        ↪ Trip_T_0\n",
    "description" :"ENFORCED: in the interval defined by the entire execution.\nTRIGGER:
        ↪ first point in the interval if <b><i>(( ! Bypass_T_0 & ( T_0 > Setpoint_T_0 ) ))
        ↪ </i></b> is true and any point in the interval where <b><i>(( ! Bypass_T_0 & (
        ↪ T_0 > Setpoint_T_0 ) ))</i></b> becomes true (from false).\nREQUIRES: for every
        ↪ trigger, RES must hold at all time points between (and including) the trigger
        ↪ and the end of the interval."
  },
  {
    "reqid" :"INSTRUMENTATION_0_T_TRIP_UI",
    "parent_reqid" :"",
    "rationale" :"RFP function 12",
    "fulltext" :"RTS shall always satisfy UI_Trip_T_0_Display = Trip_T_0\n",
    "description" :"ENFORCED: in the interval defined by the entire execution.\nTRIGGER:
        ↪ first point in the interval.\nREQUIRES: for every trigger, RES must hold at all
        ↪ time points between (and including) the trigger and the end of the interval."
  },
  {
    "reqid" :"INSTRUMENTATION_0_P_TRIP",
    "parent_reqid" :"",
    "rationale" :"RFP function 1,2,3,9\n",
    "fulltext" :"When\n (!Bypass_P_0 & (P_0 > Setpoint_P_0))\nRTS shall always satisfy
        ↪ Trip_P_0\n",
    "description" :"ENFORCED: in the interval defined by the entire execution.\nTRIGGER:
        ↪ first point in the interval if <b><i>(( ! Bypass_P_0 & ( P_0 > Setpoint_P_0 ) ))
        ↪ </i></b> is true and any point in the interval where <b><i>(( ! Bypass_P_0 & (
        ↪ P_0 > Setpoint_P_0 ) ))</i></b> becomes true (from false).\nREQUIRES: for every
        ↪ trigger, RES must hold at all time points between (and including) the trigger
        ↪ and the end of the interval."
  },
  {
    "reqid" :"INSTRUMENTATION_0_P_TRIP_UI",
    "parent_reqid" :"",
    "rationale" :"RFP function 12",
    "fulltext" :"RTS shall always satisfy UI_Trip_P_0_Display = Trip_P_0\n",
    "description" :"ENFORCED: in the interval defined by the entire execution.\nTRIGGER:
        ↪ first point in the interval.\nREQUIRES: for every trigger, RES must hold at all
        ↪ time points between (and including) the trigger and the end of the interval."
  },
  {
    "reqid" :"INSTRUMENTATION_0_S_TRIP",
    "parent_reqid" :"",
    "rationale" :"RFP function 1,2,3,9\n",
    "fulltext" :"When\n (!Bypass_S_0 & (S_0 < Setpoint_S_0))\nRTS shall always satisfy
        ↪ Trip_S_0\n",
    "description" :"ENFORCED: in the interval defined by the entire execution.\nTRIGGER:
        ↪ first point in the interval if <b><i>(( ! Bypass_S_0 & ( S_0 < Setpoint_S_0 ) ))
        ↪ </i></b> is true and any point in the interval where <b><i>(( ! Bypass_S_0 & (
        ↪ S_0 < Setpoint_S_0 ) ))</i></b> becomes true (from false).\nREQUIRES: for every
        ↪ trigger, RES must hold at all time points between (and including) the trigger
        ↪ and the end of the interval."
  },
  {
    "reqid" :"INSTRUMENTATION_0_S_TRIP_UI",
    "parent_reqid" :"",
    "rationale" :"RFP function 12",
```

```
    "fulltext" :"RTS shall always satisfy UI_Trip_S_0_Display = Trip_S_0\n",
    "description" :"ENFORCED: in the interval defined by the entire execution.\nTRIGGER:
        ↪ first point in the interval.\nREQUIRES: for every trigger, RES must hold at all
        ↪ time points between (and including) the trigger and the end of the interval."
  },
  {
    "reqid" :"INSTRUMENTATION_1_T_TRIP",
    "parent_reqid" :"",
    "rationale" :"RFP function 1,2,3,9\n",
    "fulltext" :"When\n (!Bypass_T_1 & (T_1 > Setpoint_T_1))\nRTS shall always satisfy
        ↪ Trip_T_1\n",
    "description" :"ENFORCED: in the interval defined by the entire execution.\nTRIGGER:
        ↪ first point in the interval if <b><i>(( ! Bypass_T_1 & ( T_1 > Setpoint_T_1 ) ))
        ↪ </i></b> is true and any point in the interval where <b><i>(( ! Bypass_T_1 & (
        ↪ T_1 > Setpoint_T_1 ) ))</i></b> becomes true (from false).\nREQUIRES: for every
        ↪ trigger, RES must hold at all time points between (and including) the trigger
        ↪ and the end of the interval."
  },
  {
    "reqid" :"INSTRUMENTATION_1_T_TRIP_UI",
    "parent_reqid" :"",
    "rationale" :"RFP function 12",
    "fulltext" :"RTS shall always satisfy UI_Trip_T_1_Display = Trip_T_1\n",
    "description" :"ENFORCED: in the interval defined by the entire execution.\nTRIGGER:
        ↪ first point in the interval.\nREQUIRES: for every trigger, RES must hold at all
        ↪ time points between (and including) the trigger and the end of the interval."
  },
  {
    "reqid" :"INSTRUMENTATION_1_P_TRIP",
    "parent_reqid" :"",
    "rationale" :"RFP function 1,2,3,9\n",
    "fulltext" :"When\n (!Bypass_P_1 & (P_1 > Setpoint_P_1))\nRTS shall always satisfy
        ↪ Trip_P_1\n",
    "description" :"ENFORCED: in the interval defined by the entire execution.\nTRIGGER:
        ↪ first point in the interval if <b><i>(( ! Bypass_P_1 & ( P_1 > Setpoint_P_1 ) ))
        ↪ </i></b> is true and any point in the interval where <b><i>(( ! Bypass_P_1 & (
        ↪ P_1 > Setpoint_P_1 ) ))</i></b> becomes true (from false).\nREQUIRES: for every
        ↪ trigger, RES must hold at all time points between (and including) the trigger
        ↪ and the end of the interval."
  },
  {
    "reqid" :"INSTRUMENTATION_1_P_TRIP_UI",
    "parent_reqid" :"",
    "rationale" :"RFP function 12",
    "fulltext" :"RTS shall always satisfy UI_Trip_P_1_Display = Trip_P_1\n",
    "description" :"ENFORCED: in the interval defined by the entire execution.\nTRIGGER:
        ↪ first point in the interval.\nREQUIRES: for every trigger, RES must hold at all
        ↪ time points between (and including) the trigger and the end of the interval."
  },
  {
    "reqid" :"INSTRUMENTATION_1_S_TRIP",
    "parent_reqid" :"",
    "rationale" :"RFP function 1,2,3,9\n",
    "fulltext" :"When\n (!Bypass_S_1 & (S_1 < Setpoint_S_1))\nRTS shall always satisfy
        ↪ Trip_S_1\n",
    "description" :"ENFORCED: in the interval defined by the entire execution.\nTRIGGER:
        ↪ first point in the interval if <b><i>(( ! Bypass_S_1 & ( S_1 < Setpoint_S_1 ) ))
        ↪ </i></b> is true and any point in the interval where <b><i>(( ! Bypass_S_1 & (
        ↪ S_1 < Setpoint_S_1 ) ))</i></b> becomes true (from false).\nREQUIRES: for every
        ↪ trigger, RES must hold at all time points between (and including) the trigger
        ↪ and the end of the interval."
  },
  {
    "reqid" :"INSTRUMENTATION_1_S_TRIP_UI",
    "parent_reqid" :"",
    "rationale" :"RFP function 12",
    "fulltext" :"RTS shall always satisfy UI_Trip_S_1_Display = Trip_S_1\n",
```

```
    "description" :"ENFORCED: in the interval defined by the entire execution.\nTRIGGER:
        ↪ first point in the interval.\nREQUIRES: for every trigger, RES must hold at all
        ↪ time points between (and including) the trigger and the end of the interval."
  },
  {
    "reqid" :"INSTRUMENTATION_2_T_TRIP",
    "parent_reqid" :"",
    "rationale" :"RFP function 1,2,3,9\n",
    "fulltext" :"When\n (!Bypass_T_2 & (T_2 > Setpoint_T_2))\nRTS shall always satisfy
        ↪ Trip_T_2\n",
    "description" :"ENFORCED: in the interval defined by the entire execution.\nTRIGGER:
        ↪ first point in the interval if <b><i>(( ! Bypass_T_2 & ( T_2 > Setpoint_T_2 ) ))
        ↪ </i></b> is true and any point in the interval where <b><i>(( ! Bypass_T_2 & (
        ↪ T_2 > Setpoint_T_2 ) ))</i></b> becomes true (from false).\nREQUIRES: for every
        ↪ trigger, RES must hold at all time points between (and including) the trigger
        ↪ and the end of the interval."
  },
  {
    "reqid" :"INSTRUMENTATION_2_T_TRIP_UI",
    "parent_reqid" :"",
    "rationale" :"RFP function 12",
    "fulltext" :"RTS shall always satisfy UI_Trip_T_2_Display = Trip_T_2\n",
    "description" :"ENFORCED: in the interval defined by the entire execution.\nTRIGGER:
        ↪ first point in the interval.\nREQUIRES: for every trigger, RES must hold at all
        ↪ time points between (and including) the trigger and the end of the interval."
  },
  {
    "reqid" :"INSTRUMENTATION_2_P_TRIP",
    "parent_reqid" :"",
    "rationale" :"RFP function 1,2,3,9\n",
    "fulltext" :"When\n (!Bypass_P_2 & (P_2 > Setpoint_P_2))\nRTS shall always satisfy
        ↪ Trip_P_2\n",
    "description" :"ENFORCED: in the interval defined by the entire execution.\nTRIGGER:
        ↪ first point in the interval if <b><i>(( ! Bypass_P_2 & ( P_2 > Setpoint_P_2 ) ))
        ↪ </i></b> is true and any point in the interval where <b><i>(( ! Bypass_P_2 & (
        ↪ P_2 > Setpoint_P_2 ) ))</i></b> becomes true (from false).\nREQUIRES: for every
        ↪ trigger, RES must hold at all time points between (and including) the trigger
        ↪ and the end of the interval."
  },
  {
    "reqid" :"INSTRUMENTATION_2_P_TRIP_UI",
    "parent_reqid" :"",
    "rationale" :"RFP function 12",
    "fulltext" :"RTS shall always satisfy UI_Trip_P_2_Display = Trip_P_2\n",
    "description" :"ENFORCED: in the interval defined by the entire execution.\nTRIGGER:
        ↪ first point in the interval.\nREQUIRES: for every trigger, RES must hold at all
        ↪ time points between (and including) the trigger and the end of the interval."
  },
  {
    "reqid" :"INSTRUMENTATION_2_S_TRIP",
    "parent_reqid" :"",
    "rationale" :"RFP function 1,2,3,9\n",
    "fulltext" :"When\n (!Bypass_S_2 & (S_2 < Setpoint_S_2))\nRTS shall always satisfy
        ↪ Trip_S_2\n",
    "description" :"ENFORCED: in the interval defined by the entire execution.\nTRIGGER:
        ↪ first point in the interval if <b><i>(( ! Bypass_S_2 & ( S_2 < Setpoint_S_2 ) ))
        ↪ </i></b> is true and any point in the interval where <b><i>(( ! Bypass_S_2 & (
        ↪ S_2 < Setpoint_S_2 ) ))</i></b> becomes true (from false).\nREQUIRES: for every
        ↪ trigger, RES must hold at all time points between (and including) the trigger
        ↪ and the end of the interval."
  },
  {
    "reqid" :"INSTRUMENTATION_2_S_TRIP_UI",
    "parent_reqid" :"",
    "rationale" :"RFP function 12",
    "fulltext" :"RTS shall always satisfy UI_Trip_S_2_Display = Trip_S_2\n",
    "description" :"ENFORCED: in the interval defined by the entire execution.\nTRIGGER:
        ↪ first point in the interval.\nREQUIRES: for every trigger, RES must hold at all
```

```
        ↪ time points between (and including) the trigger and the end of the interval."
  },
  {
    "reqid" :"INSTRUMENTATION_3_T_TRIP",
    "parent_reqid" :"",
    "rationale" :"RFP function 1,2,3,9\n",
    "fulltext" :"When\n (!Bypass_T_3 & (T_3 > Setpoint_T_3))\nRTS shall always satisfy
        ↪ Trip_T_3\n",
    "description" :"ENFORCED: in the interval defined by the entire execution.\nTRIGGER:
        ↪ first point in the interval if <b><i>(( ! Bypass_T_3 & ( T_3 > Setpoint_T_3 ) ))
        ↪ </i></b> is true and any point in the interval where <b><i>(( ! Bypass_T_3 & (
        ↪ T_3 > Setpoint_T_3 ) ))</i></b> becomes true (from false).\nREQUIRES: for every
        ↪ trigger, RES must hold at all time points between (and including) the trigger
        ↪ and the end of the interval."
  },
  {
    "reqid" :"INSTRUMENTATION_3_T_TRIP_UI",
    "parent_reqid" :"",
    "rationale" :"RFP function 12",
    "fulltext" :"RTS shall always satisfy UI_Trip_T_3_Display = Trip_T_3\n",
    "description" :"ENFORCED: in the interval defined by the entire execution.\nTRIGGER:
        ↪ first point in the interval.\nREQUIRES: for every trigger, RES must hold at all
        ↪ time points between (and including) the trigger and the end of the interval."
  },
  {
    "reqid" :"INSTRUMENTATION_3_P_TRIP",
    "parent_reqid" :"",
    "rationale" :"RFP function 1,2,3,9\n",
    "fulltext" :"When\n (!Bypass_P_3 & (P_3 > Setpoint_P_3))\nRTS shall always satisfy
        ↪ Trip_P_3\n",
    "description" :"ENFORCED: in the interval defined by the entire execution.\nTRIGGER:
        ↪ first point in the interval if <b><i>(( ! Bypass_P_3 & ( P_3 > Setpoint_P_3 ) ))
        ↪ </i></b> is true and any point in the interval where <b><i>(( ! Bypass_P_3 & (
        ↪ P_3 > Setpoint_P_3 ) ))</i></b> becomes true (from false).\nREQUIRES: for every
        ↪ trigger, RES must hold at all time points between (and including) the trigger
        ↪ and the end of the interval."
  },
  {
    "reqid" :"INSTRUMENTATION_3_P_TRIP_UI",
    "parent_reqid" :"",
    "rationale" :"RFP function 12",
    "fulltext" :"RTS shall always satisfy UI_Trip_P_3_Display = Trip_P_3\n",
    "description" :"ENFORCED: in the interval defined by the entire execution.\nTRIGGER:
        ↪ first point in the interval.\nREQUIRES: for every trigger, RES must hold at all
        ↪ time points between (and including) the trigger and the end of the interval."
  },
  {
    "reqid" :"INSTRUMENTATION_3_S_TRIP",
    "parent_reqid" :"",
    "rationale" :"RFP function 1,2,3,9\n",
    "fulltext" :"When\n (!Bypass_S_3 & (S_3 < Setpoint_S_3))\nRTS shall always satisfy
        ↪ Trip_S_3\n",
    "description" :"ENFORCED: in the interval defined by the entire execution.\nTRIGGER:
        ↪ first point in the interval if <b><i>(( ! Bypass_S_3 & ( S_3 < Setpoint_S_3 ) ))
        ↪ </i></b> is true and any point in the interval where <b><i>(( ! Bypass_S_3 & (
        ↪ S_3 < Setpoint_S_3 ) ))</i></b> becomes true (from false).\nREQUIRES: for every
        ↪ trigger, RES must hold at all time points between (and including) the trigger
        ↪ and the end of the interval."
  },
  {
    "reqid" :"INSTRUMENTATION_3_S_TRIP_UI",
    "parent_reqid" :"",
    "rationale" :"RFP function 12",
    "fulltext" :"RTS shall always satisfy UI_Trip_S_3_Display = Trip_S_3\n",
    "description" :"ENFORCED: in the interval defined by the entire execution.\nTRIGGER:
        ↪ first point in the interval.\nREQUIRES: for every trigger, RES must hold at all
        ↪ time points between (and including) the trigger and the end of the interval."
  },
```

```
254   {
255     "reqid" :"INSTRUMENTATION_0_BYPASS_T_Display",
256     "parent_reqid" :"",
257     "rationale" :"RFP function 13",
258     "fulltext" :"RTS shall always satisfy UI_Bypass_T_0_Display = Bypass_T_0\n",
259     "description" :"ENFORCED: in the interval defined by the entire execution.\nTRIGGER:
          ↪ first point in the interval.\nREQUIRES: for every trigger, RES must hold at all
          ↪ time points between (and including) the trigger and the end of the interval."
260   },
261   {
262     "reqid" :"INSTRUMENTATION_0_BYPASS_T",
263     "parent_reqid" :"",
264     "rationale" :"RFP function 9:configure in maintenance mode to bypass an instrument
          ↪ channel (this requirement sets the system value from the UI)\n",
265     "fulltext" :"RTS shall always satisfy\n if UI_Maintenance_0\n then (UI_Bypass_T_0) =
          ↪ Bypass_T_0)\n",
266     "description" :"ENFORCED: in the interval defined by the entire execution.\nTRIGGER:
          ↪ first point in the interval.\nREQUIRES: for every trigger, RES must hold at all
          ↪ time points between (and including) the trigger and the end of the interval."
267   },
268   {
269     "reqid" :"CONST_INSTRUMENTATION_0_NO_BYPASS_T",
270     "parent_reqid" :"INSTRUMENTATION_0_BYPASS_T",
271     "rationale" :"RFP function 9:configure in maintenance mode to bypass an instrument
          ↪ channel (this requirement does not allow bypass to turn on during normal
          ↪ operation)\n",
272     "fulltext" :"When !UI_Maintenance_0 & !Bypass_T_0 RTS shall, until UI_Maintenance_0 &
          ↪ UI_Bypass_T_0, satisfy\n !Bypass_T_0\n",
273     "description" :"ENFORCED: in the interval defined by the entire execution.\nTRIGGER:
          ↪ first point in the interval if <b><i>(! UI_Maintenance_0 & ! Bypass_T_0)</i></b>
          ↪  is true and any point in the interval where <b><i>(! UI_Maintenance_0 & !
          ↪ Bypass_T_0)</i></b> becomes true (from false).\nREQUIRES: for every trigger, RES
          ↪  must remain true until (but not necessarily including) the point where the stop
          ↪  condition holds, or to the end of the interval. If the stop condition never
          ↪ occurs, RES must hold until the end of the scope, or forever. If the stop
          ↪ condition holds at the trigger, the requirement is satisfied."
274   },
275   {
276     "reqid" :"CONST_INSTRUMENTATION_0_BYPASS_T",
277     "parent_reqid" :"INSTRUMENTATION_0_BYPASS_T",
278     "rationale" :"RFP function 9:configure in maintenance mode to bypass an instrument
          ↪ channel (this requirement does not allow bypass to turn off during normal
          ↪ operation)\n",
279     "fulltext" :"When !UI_Maintenance_0 & Bypass_T_0 RTS shall, until UI_Maintenance_0 & !
          ↪ UI_Bypass_T_0, satisfy\n Bypass_T_0\n",
280     "description" :"ENFORCED: in the interval defined by the entire execution.\nTRIGGER:
          ↪ first point in the interval if <b><i>(! UI_Maintenance_0 & Bypass_T_0)</i></b>
          ↪ is true and any point in the interval where <b><i>(! UI_Maintenance_0 &
          ↪ Bypass_T_0)</i></b> becomes true (from false).\nREQUIRES: for every trigger, RES
          ↪  must remain true until (but not necessarily including) the point where the stop
          ↪  condition holds, or to the end of the interval. If the stop condition never
          ↪ occurs, RES must hold until the end of the scope, or forever. If the stop
          ↪ condition holds at the trigger, the requirement is satisfied."
281   },
282   {
283     "reqid" :"INSTRUMENTATION_0_BYPASS_P_Display",
284     "parent_reqid" :"",
285     "rationale" :"RFP function 13",
286     "fulltext" :"RTS shall always satisfy UI_Bypass_P_0_Display = Bypass_P_0\n",
287     "description" :"ENFORCED: in the interval defined by the entire execution.\nTRIGGER:
          ↪ first point in the interval.\nREQUIRES: for every trigger, RES must hold at all
          ↪ time points between (and including) the trigger and the end of the interval."
288   },
289   {
290     "reqid" :"INSTRUMENTATION_0_BYPASS_P",
291     "parent_reqid" :"",
292     "rationale" :"RFP function 9:configure in maintenance mode to bypass an instrument
          ↪ channel (this requirement sets the system value from the UI)\n",
```

```
    "fulltext" :"RTS shall always satisfy\n if UI_Maintenance_0\n then (UI_Bypass_P_0) =
        ↪ Bypass_P_0)\n",
    "description" :"ENFORCED: in the interval defined by the entire execution.\nTRIGGER:
        ↪ first point in the interval.\nREQUIRES: for every trigger, RES must hold at all
        ↪ time points between (and including) the trigger and the end of the interval."
  },
  {
    "reqid" :"CONST_INSTRUMENTATION_0_NO_BYPASS_P",
    "parent_reqid" :"INSTRUMENTATION_0_BYPASS_P",
    "rationale" :"RFP function 9:configure in maintenance mode to bypass an instrument
        ↪ channel (this requirement does not allow bypass to turn on during normal
        ↪ operation)\n",
    "fulltext" :"When !UI_Maintenance_0 & !Bypass_P_0 RTS shall, until UI_Maintenance_0 &
        ↪ UI_Bypass_P_0, satisfy\n !Bypass_P_0\n",
    "description" :"ENFORCED: in the interval defined by the entire execution.\nTRIGGER:
        ↪ first point in the interval if <b><i>(! UI_Maintenance_0 & ! Bypass_P_0)</i></b>
        ↪  is true and any point in the interval where <b><i>(! UI_Maintenance_0 & !
        ↪ Bypass_P_0)</i></b> becomes true (from false).\nREQUIRES: for every trigger, RES
        ↪  must remain true until (but not necessarily including) the point where the stop
        ↪  condition holds, or to the end of the interval. If the stop condition never
        ↪ occurs, RES must hold until the end of the scope, or forever. If the stop
        ↪ condition holds at the trigger, the requirement is satisfied."
  },
  {
    "reqid" :"CONST_INSTRUMENTATION_0_BYPASS_P",
    "parent_reqid" :"INSTRUMENTATION_0_BYPASS_P",
    "rationale" :"RFP function 9:configure in maintenance mode to bypass an instrument
        ↪ channel (this requirement does not allow bypass to turn off during normal
        ↪ operation)\n",
    "fulltext" :"When !UI_Maintenance_0 & Bypass_P_0 RTS shall, until UI_Maintenance_0 & !
        ↪ UI_Bypass_P_0, satisfy\n Bypass_P_0\n",
    "description" :"ENFORCED: in the interval defined by the entire execution.\nTRIGGER:
        ↪ first point in the interval if <b><i>(! UI_Maintenance_0 & Bypass_P_0)</i></b>
        ↪ is true and any point in the interval where <b><i>(! UI_Maintenance_0 &
        ↪ Bypass_P_0)</i></b> becomes true (from false).\nREQUIRES: for every trigger, RES
        ↪  must remain true until (but not necessarily including) the point where the stop
        ↪  condition holds, or to the end of the interval. If the stop condition never
        ↪ occurs, RES must hold until the end of the scope, or forever. If the stop
        ↪ condition holds at the trigger, the requirement is satisfied."
  },
  {
    "reqid" :"INSTRUMENTATION_UI_T_0",
    "parent_reqid" :"",
    "rationale" :"RFP function 11",
    "fulltext" :"RTS shall always satisfy T_0 = UI_T_0_Display",
    "description" :"ENFORCED: in the interval defined by the entire execution.\nTRIGGER:
        ↪ first point in the interval.\nREQUIRES: for every trigger, RES must hold at all
        ↪ time points between (and including) the trigger and the end of the interval."
  },
  {
    "reqid" :"INSTRUMENTATION_0_SETPOINT_T",
    "parent_reqid" :"",
    "rationale" :"RFP function 8",
    "fulltext" :"RTS shall always satisfy\n if UI_Maintenance_0 then Setpoint_T_0 =
        ↪ UI_Setpoint_T_0\n",
    "description" :"ENFORCED: in the interval defined by the entire execution.\nTRIGGER:
        ↪ first point in the interval.\nREQUIRES: for every trigger, RES must hold at all
        ↪ time points between (and including) the trigger and the end of the interval."
  },
  {
    "reqid" :"INSTRUMENTATION_0_MANUAL_TRIP_T",
    "parent_reqid" :"",
    "rationale" :"RFP function 10",
    "fulltext" :"RTS shall always satisfy\n if UI_Maintenance_0 & UI_Manual_Trip_T_0 then
        ↪ Trip_T_0\n",
    "description" :"ENFORCED: in the interval defined by the entire execution.\nTRIGGER:
        ↪ first point in the interval.\nREQUIRES: for every trigger, RES must hold at all
        ↪ time points between (and including) the trigger and the end of the interval."
```

```
330   },
331   {
332     "reqid" :"INSTRUMENTATION_UI_P_0",
333     "parent_reqid" :"",
334     "rationale" :"RFP function 11",
335     "fulltext" :"RTS shall always satisfy P_0 = UI_P_0_Display",
336     "description" :"ENFORCED: in the interval defined by the entire execution.\nTRIGGER:
          ↪ first point in the interval.\nREQUIRES: for every trigger, RES must hold at all
          ↪ time points between (and including) the trigger and the end of the interval."
337   },
338   {
339     "reqid" :"INSTRUMENTATION_0_SETPOINT_P",
340     "parent_reqid" :"",
341     "rationale" :"RFP function 8",
342     "fulltext" :"RTS shall always satisfy\n if UI_Maintenance_0 then Setpoint_P_0 =
          ↪ UI_Setpoint_P_0\n",
343     "description" :"ENFORCED: in the interval defined by the entire execution.\nTRIGGER:
          ↪ first point in the interval.\nREQUIRES: for every trigger, RES must hold at all
          ↪ time points between (and including) the trigger and the end of the interval."
344   },
345   {
346     "reqid" :"INSTRUMENTATION_0_MANUAL_TRIP_P",
347     "parent_reqid" :"",
348     "rationale" :"RFP function 10",
349     "fulltext" :"RTS shall always satisfy\n if UI_Maintenance_0 & UI_Manual_Trip_P_0 then
          ↪ Trip_P_0\n",
350     "description" :"ENFORCED: in the interval defined by the entire execution.\nTRIGGER:
          ↪ first point in the interval.\nREQUIRES: for every trigger, RES must hold at all
          ↪ time points between (and including) the trigger and the end of the interval."
351   },
352   {
353     "reqid" :"INSTRUMENTATION_UI_S_0",
354     "parent_reqid" :"",
355     "rationale" :"RFP function 11",
356     "fulltext" :"RTS shall always satisfy S_0 = UI_S_0_Display",
357     "description" :"ENFORCED: in the interval defined by the entire execution.\nTRIGGER:
          ↪ first point in the interval.\nREQUIRES: for every trigger, RES must hold at all
          ↪ time points between (and including) the trigger and the end of the interval."
358   },
359   {
360     "reqid" :"INSTRUMENTATION_0_SETPOINT_S",
361     "parent_reqid" :"",
362     "rationale" :"RFP function 8",
363     "fulltext" :"RTS shall always satisfy\n if UI_Maintenance_0 then Setpoint_S_0 =
          ↪ UI_Setpoint_S_0\n",
364     "description" :"ENFORCED: in the interval defined by the entire execution.\nTRIGGER:
          ↪ first point in the interval.\nREQUIRES: for every trigger, RES must hold at all
          ↪ time points between (and including) the trigger and the end of the interval."
365   },
366   {
367     "reqid" :"INSTRUMENTATION_0_MANUAL_TRIP_S",
368     "parent_reqid" :"",
369     "rationale" :"RFP function 10",
370     "fulltext" :"RTS shall always satisfy\n if UI_Maintenance_0 & UI_Manual_Trip_S_0 then
          ↪ Trip_S_0\n",
371     "description" :"ENFORCED: in the interval defined by the entire execution.\nTRIGGER:
          ↪ first point in the interval.\nREQUIRES: for every trigger, RES must hold at all
          ↪ time points between (and including) the trigger and the end of the interval."
372   },
373   {
374     "reqid" :"INSTRUMENTATION_1_BYPASS_T_Display",
375     "parent_reqid" :"",
376     "rationale" :"RFP function 13",
377     "fulltext" :"RTS shall always satisfy UI_Bypass_T_1_Display = Bypass_T_1\n",
378     "description" :"ENFORCED: in the interval defined by the entire execution.\nTRIGGER:
          ↪ first point in the interval.\nREQUIRES: for every trigger, RES must hold at all
          ↪ time points between (and including) the trigger and the end of the interval."
379   },
```

```
{
  "reqid" :"INSTRUMENTATION_1_BYPASS_T",
  "parent_reqid" :"",
  "rationale" :"RFP function 9:configure in maintenance mode to bypass an instrument channel (this requirement sets the system value from the UI)\n",
  "fulltext" :"RTS shall always satisfy\n if UI_Maintenance_1\n then (UI_Bypass_T_1) = Bypass_T_1)\n",
  "description" :"ENFORCED: in the interval defined by the entire execution.\nTRIGGER: first point in the interval.\nREQUIRES: for every trigger, RES must hold at all time points between (and including) the trigger and the end of the interval."
},
{
  "reqid" :"CONST_INSTRUMENTATION_1_NO_BYPASS_T",
  "parent_reqid" :"INSTRUMENTATION_1_BYPASS_T",
  "rationale" :"RFP function 9:configure in maintenance mode to bypass an instrument channel (this requirement does not allow bypass to turn on during normal operation)\n",
  "fulltext" :"When !UI_Maintenance_1 & !Bypass_T_1 RTS shall, until UI_Maintenance_1 & UI_Bypass_T_1, satisfy\n !Bypass_T_1\n",
  "description" :"ENFORCED: in the interval defined by the entire execution.\nTRIGGER: first point in the interval if <b><i>(! UI_Maintenance_1 & ! Bypass_T_1)</i></b> is true and any point in the interval where <b><i>(! UI_Maintenance_1 & ! Bypass_T_1)</i></b> becomes true (from false).\nREQUIRES: for every trigger, RES must remain true until (but not necessarily including) the point where the stop condition holds, or to the end of the interval. If the stop condition never occurs, RES must hold until the end of the scope, or forever. If the stop condition holds at the trigger, the requirement is satisfied."
},
{
  "reqid" :"CONST_INSTRUMENTATION_1_BYPASS_T",
  "parent_reqid" :"INSTRUMENTATION_1_BYPASS_T",
  "rationale" :"RFP function 9:configure in maintenance mode to bypass an instrument channel (this requirement does not allow bypass to turn off during normal operation)\n",
  "fulltext" :"When !UI_Maintenance_1 & Bypass_T_1 RTS shall, until UI_Maintenance_1 & ! UI_Bypass_T_1, satisfy\n Bypass_T_1\n",
  "description" :"ENFORCED: in the interval defined by the entire execution.\nTRIGGER: first point in the interval if <b><i>(! UI_Maintenance_1 & Bypass_T_1)</i></b> is true and any point in the interval where <b><i>(! UI_Maintenance_1 & Bypass_T_1)</i></b> becomes true (from false).\nREQUIRES: for every trigger, RES must remain true until (but not necessarily including) the point where the stop condition holds, or to the end of the interval. If the stop condition never occurs, RES must hold until the end of the scope, or forever. If the stop condition holds at the trigger, the requirement is satisfied."
},
{
  "reqid" :"INSTRUMENTATION_1_BYPASS_P_Display",
  "parent_reqid" :"",
  "rationale" :"RFP function 13",
  "fulltext" :"RTS shall always satisfy UI_Bypass_P_1_Display = Bypass_P_1\n",
  "description" :"ENFORCED: in the interval defined by the entire execution.\nTRIGGER: first point in the interval.\nREQUIRES: for every trigger, RES must hold at all time points between (and including) the trigger and the end of the interval."
},
{
  "reqid" :"INSTRUMENTATION_1_BYPASS_P",
  "parent_reqid" :"",
  "rationale" :"RFP function 9:configure in maintenance mode to bypass an instrument channel (this requirement sets the system value from the UI)\n",
  "fulltext" :"RTS shall always satisfy\n if UI_Maintenance_1\n then (UI_Bypass_P_1) = Bypass_P_1)\n",
  "description" :"ENFORCED: in the interval defined by the entire execution.\nTRIGGER: first point in the interval.\nREQUIRES: for every trigger, RES must hold at all time points between (and including) the trigger and the end of the interval."
},
{
  "reqid" :"CONST_INSTRUMENTATION_1_NO_BYPASS_P",
  "parent_reqid" :"INSTRUMENTATION_1_BYPASS_P",
```

```
418     "rationale" :"RFP function 9:configure in maintenance mode to bypass an instrument
            ↪ channel (this requirement does not allow bypass to turn on during normal
            ↪ operation)\n",
419     "fulltext" :"When !UI_Maintenance_1 & !Bypass_P_1 RTS shall, until UI_Maintenance_1 &
            ↪ UI_Bypass_P_1, satisfy\n !Bypass_P_1\n",
420     "description" :"ENFORCED: in the interval defined by the entire execution.\nTRIGGER:
            ↪ first point in the interval if <b><i>(! UI_Maintenance_1 & ! Bypass_P_1)</i></b>
            ↪  is true and any point in the interval where <b><i>(! UI_Maintenance_1 & !
            ↪ Bypass_P_1)</i></b> becomes true (from false).\nREQUIRES: for every trigger, RES
            ↪  must remain true until (but not necessarily including) the point where the stop
            ↪  condition holds, or to the end of the interval. If the stop condition never
            ↪ occurs, RES must hold until the end of the scope, or forever. If the stop
            ↪ condition holds at the trigger, the requirement is satisfied."
421   },
422   {
423     "reqid" :"CONST_INSTRUMENTATION_1_BYPASS_P",
424     "parent_reqid" :"INSTRUMENTATION_1_BYPASS_P",
425     "rationale" :"RFP function 9:configure in maintenance mode to bypass an instrument
            ↪ channel (this requirement does not allow bypass to turn off during normal
            ↪ operation)\n",
426     "fulltext" :"When !UI_Maintenance_1 & Bypass_P_1 RTS shall, until UI_Maintenance_1 & !
            ↪ UI_Bypass_P_1, satisfy\n Bypass_P_1\n",
427     "description" :"ENFORCED: in the interval defined by the entire execution.\nTRIGGER:
            ↪ first point in the interval if <b><i>(! UI_Maintenance_1 & Bypass_P_1)</i></b>
            ↪ is true and any point in the interval where <b><i>(! UI_Maintenance_1 &
            ↪ Bypass_P_1)</i></b> becomes true (from false).\nREQUIRES: for every trigger, RES
            ↪  must remain true until (but not necessarily including) the point where the stop
            ↪  condition holds, or to the end of the interval. If the stop condition never
            ↪ occurs, RES must hold until the end of the scope, or forever. If the stop
            ↪ condition holds at the trigger, the requirement is satisfied."
428   },
429   {
430     "reqid" :"INSTRUMENTATION_UI_T_1",
431     "parent_reqid" :"",
432     "rationale" :"RFP function 11",
433     "fulltext" :"RTS shall always satisfy T_1 = UI_T_1_Display",
434     "description" :"ENFORCED: in the interval defined by the entire execution.\nTRIGGER:
            ↪ first point in the interval.\nREQUIRES: for every trigger, RES must hold at all
            ↪ time points between (and including) the trigger and the end of the interval."
435   },
436   {
437     "reqid" :"INSTRUMENTATION_1_SETPOINT_T",
438     "parent_reqid" :"",
439     "rationale" :"RFP function 8",
440     "fulltext" :"RTS shall always satisfy\n if UI_Maintenance_1 then Setpoint_T_1 =
            ↪ UI_Setpoint_T_1\n",
441     "description" :"ENFORCED: in the interval defined by the entire execution.\nTRIGGER:
            ↪ first point in the interval.\nREQUIRES: for every trigger, RES must hold at all
            ↪ time points between (and including) the trigger and the end of the interval."
442   },
443   {
444     "reqid" :"INSTRUMENTATION_1_MANUAL_TRIP_T",
445     "parent_reqid" :"",
446     "rationale" :"RFP function 10",
447     "fulltext" :"RTS shall always satisfy\n if UI_Maintenance_1 & UI_Manual_Trip_T_1 then
            ↪ Trip_T_1\n",
448     "description" :"ENFORCED: in the interval defined by the entire execution.\nTRIGGER:
            ↪ first point in the interval.\nREQUIRES: for every trigger, RES must hold at all
            ↪ time points between (and including) the trigger and the end of the interval."
449   },
450   {
451     "reqid" :"INSTRUMENTATION_UI_P_1",
452     "parent_reqid" :"",
453     "rationale" :"RFP function 11",
454     "fulltext" :"RTS shall always satisfy P_1 = UI_P_1_Display",
455     "description" :"ENFORCED: in the interval defined by the entire execution.\nTRIGGER:
            ↪ first point in the interval.\nREQUIRES: for every trigger, RES must hold at all
            ↪ time points between (and including) the trigger and the end of the interval."
```

```
  },
  {
    "reqid" :"INSTRUMENTATION_1_SETPOINT_P",
    "parent_reqid" :"",
    "rationale" :"RFP function 8",
    "fulltext" :"RTS shall always satisfy\n if UI_Maintenance_1 then Setpoint_P_1 =
        ↪ UI_Setpoint_P_1\n",
    "description" :"ENFORCED: in the interval defined by the entire execution.\nTRIGGER:
        ↪ first point in the interval.\nREQUIRES: for every trigger, RES must hold at all
        ↪ time points between (and including) the trigger and the end of the interval."
  },
  {
    "reqid" :"INSTRUMENTATION_1_MANUAL_TRIP_P",
    "parent_reqid" :"",
    "rationale" :"RFP function 10",
    "fulltext" :"RTS shall always satisfy\n if UI_Maintenance_1 & UI_Manual_Trip_P_1 then
        ↪ Trip_P_1\n",
    "description" :"ENFORCED: in the interval defined by the entire execution.\nTRIGGER:
        ↪ first point in the interval.\nREQUIRES: for every trigger, RES must hold at all
        ↪ time points between (and including) the trigger and the end of the interval."
  },
  {
    "reqid" :"INSTRUMENTATION_UI_S_1",
    "parent_reqid" :"",
    "rationale" :"RFP function 11",
    "fulltext" :"RTS shall always satisfy S_1 = UI_S_1_Display",
    "description" :"ENFORCED: in the interval defined by the entire execution.\nTRIGGER:
        ↪ first point in the interval.\nREQUIRES: for every trigger, RES must hold at all
        ↪ time points between (and including) the trigger and the end of the interval."
  },
  {
    "reqid" :"INSTRUMENTATION_1_SETPOINT_S",
    "parent_reqid" :"",
    "rationale" :"RFP function 8",
    "fulltext" :"RTS shall always satisfy\n if UI_Maintenance_1 then Setpoint_S_1 =
        ↪ UI_Setpoint_S_1\n",
    "description" :"ENFORCED: in the interval defined by the entire execution.\nTRIGGER:
        ↪ first point in the interval.\nREQUIRES: for every trigger, RES must hold at all
        ↪ time points between (and including) the trigger and the end of the interval."
  },
  {
    "reqid" :"INSTRUMENTATION_1_MANUAL_TRIP_S",
    "parent_reqid" :"",
    "rationale" :"RFP function 10",
    "fulltext" :"RTS shall always satisfy\n if UI_Maintenance_1 & UI_Manual_Trip_S_1 then
        ↪ Trip_S_1\n",
    "description" :"ENFORCED: in the interval defined by the entire execution.\nTRIGGER:
        ↪ first point in the interval.\nREQUIRES: for every trigger, RES must hold at all
        ↪ time points between (and including) the trigger and the end of the interval."
  },
  {
    "reqid" :"INSTRUMENTATION_2_BYPASS_T_Display",
    "parent_reqid" :"",
    "rationale" :"RFP function 13",
    "fulltext" :"RTS shall always satisfy UI_Bypass_T_2_Display = Bypass_T_2\n",
    "description" :"ENFORCED: in the interval defined by the entire execution.\nTRIGGER:
        ↪ first point in the interval.\nREQUIRES: for every trigger, RES must hold at all
        ↪ time points between (and including) the trigger and the end of the interval."
  },
  {
    "reqid" :"INSTRUMENTATION_2_BYPASS_T",
    "parent_reqid" :"",
    "rationale" :"RFP function 9:configure in maintenance mode to bypass an instrument
        ↪ channel (this requirement sets the system value from the UI)\n",
    "fulltext" :"RTS shall always satisfy\n if UI_Maintenance_2\n then (UI_Bypass_T_2) =
        ↪ Bypass_T_2)\n",
    "description" :"ENFORCED: in the interval defined by the entire execution.\nTRIGGER:
        ↪ first point in the interval.\nREQUIRES: for every trigger, RES must hold at all
```

```
        ↪ time points between (and including) the trigger and the end of the interval."
  },
  {
    "reqid" :"CONST_INSTRUMENTATION_2_NO_BYPASS_T",
    "parent_reqid" :"INSTRUMENTATION_2_BYPASS_T",
    "rationale" :"RFP function 9:configure in maintenance mode to bypass an instrument
        ↪ channel (this requirement does not allow bypass to turn on during normal
        ↪ operation)\n",
    "fulltext" :"When !UI_Maintenance_2 & !Bypass_T_2 RTS shall, until UI_Maintenance_2 &
        ↪ UI_Bypass_T_2, satisfy\n !Bypass_T_2\n",
    "description" :"ENFORCED: in the interval defined by the entire execution.\nTRIGGER:
        ↪ first point in the interval if <b><i>(! UI_Maintenance_2 & ! Bypass_T_2)</i></b>
        ↪  is true and any point in the interval where <b><i>(! UI_Maintenance_2 & !
        ↪ Bypass_T_2)</i></b> becomes true (from false).\nREQUIRES: for every trigger, RES
        ↪  must remain true until (but not necessarily including) the point where the stop
        ↪  condition holds, or to the end of the interval. If the stop condition never
        ↪ occurs, RES must hold until the end of the scope, or forever. If the stop
        ↪ condition holds at the trigger, the requirement is satisfied."
  },
  {
    "reqid" :"CONST_INSTRUMENTATION_2_BYPASS_T",
    "parent_reqid" :"INSTRUMENTATION_2_BYPASS_T",
    "rationale" :"RFP function 9:configure in maintenance mode to bypass an instrument
        ↪ channel (this requirement does not allow bypass to turn off during normal
        ↪ operation)\n",
    "fulltext" :"When !UI_Maintenance_2 & Bypass_T_2 RTS shall, until UI_Maintenance_2 & !
        ↪ UI_Bypass_T_2, satisfy\n Bypass_T_2\n",
    "description" :"ENFORCED: in the interval defined by the entire execution.\nTRIGGER:
        ↪ first point in the interval if <b><i>(! UI_Maintenance_2 & Bypass_T_2)</i></b>
        ↪ is true and any point in the interval where <b><i>(! UI_Maintenance_2 &
        ↪ Bypass_T_2)</i></b> becomes true (from false).\nREQUIRES: for every trigger, RES
        ↪  must remain true until (but not necessarily including) the point where the stop
        ↪  condition holds, or to the end of the interval. If the stop condition never
        ↪ occurs, RES must hold until the end of the scope, or forever. If the stop
        ↪ condition holds at the trigger, the requirement is satisfied."
  },
  {
    "reqid" :"INSTRUMENTATION_2_BYPASS_P_Display",
    "parent_reqid" :"",
    "rationale" :"RFP function 13",
    "fulltext" :"RTS shall always satisfy UI_Bypass_P_2_Display = Bypass_P_2\n",
    "description" :"ENFORCED: in the interval defined by the entire execution.\nTRIGGER:
        ↪ first point in the interval.\nREQUIRES: for every trigger, RES must hold at all
        ↪ time points between (and including) the trigger and the end of the interval."
  },
  {
    "reqid" :"INSTRUMENTATION_2_BYPASS_P",
    "parent_reqid" :"",
    "rationale" :"RFP function 9:configure in maintenance mode to bypass an instrument
        ↪ channel (this requirement sets the system value from the UI)\n",
    "fulltext" :"RTS shall always satisfy\n if UI_Maintenance_2\n then (UI_Bypass_P_2) =
        ↪ Bypass_P_2)\n",
    "description" :"ENFORCED: in the interval defined by the entire execution.\nTRIGGER:
        ↪ first point in the interval.\nREQUIRES: for every trigger, RES must hold at all
        ↪ time points between (and including) the trigger and the end of the interval."
  },
  {
    "reqid" :"CONST_INSTRUMENTATION_2_NO_BYPASS_P",
    "parent_reqid" :"INSTRUMENTATION_2_BYPASS_P",
    "rationale" :"RFP function 9:configure in maintenance mode to bypass an instrument
        ↪ channel (this requirement does not allow bypass to turn on during normal
        ↪ operation)\n",
    "fulltext" :"When !UI_Maintenance_2 & !Bypass_P_2 RTS shall, until UI_Maintenance_2 &
        ↪ UI_Bypass_P_2, satisfy\n !Bypass_P_2\n",
    "description" :"ENFORCED: in the interval defined by the entire execution.\nTRIGGER:
        ↪ first point in the interval if <b><i>(! UI_Maintenance_2 & ! Bypass_P_2)</i></b>
        ↪  is true and any point in the interval where <b><i>(! UI_Maintenance_2 & !
        ↪ Bypass_P_2)</i></b> becomes true (from false).\nREQUIRES: for every trigger, RES
```

```
        ↪  must remain true until (but not necessarily including) the point where the stop
        ↪  condition holds, or to the end of the interval. If the stop condition never
        ↪ occurs, RES must hold until the end of the scope, or forever. If the stop
        ↪ condition holds at the trigger, the requirement is satisfied."
  },
  {
    "reqid" :"CONST_INSTRUMENTATION_2_BYPASS_P",
    "parent_reqid" :"INSTRUMENTATION_2_BYPASS_P",
    "rationale" :"RFP function 9:configure in maintenance mode to bypass an instrument
        ↪ channel (this requirement does not allow bypass to turn off during normal
        ↪ operation)\n",
    "fulltext" :"When !UI_Maintenance_2 & Bypass_P_2 RTS shall, until UI_Maintenance_2 & !
        ↪ UI_Bypass_P_2, satisfy\n Bypass_P_2\n",
    "description" :"ENFORCED: in the interval defined by the entire execution.\nTRIGGER:
        ↪ first point in the interval if <b><i>(! UI_Maintenance_2 & Bypass_P_2)</i></b>
        ↪ is true and any point in the interval where <b><i>(! UI_Maintenance_2 &
        ↪ Bypass_P_2)</i></b> becomes true (from false).\nREQUIRES: for every trigger, RES
        ↪  must remain true until (but not necessarily including) the point where the stop
        ↪  condition holds, or to the end of the interval. If the stop condition never
        ↪ occurs, RES must hold until the end of the scope, or forever. If the stop
        ↪ condition holds at the trigger, the requirement is satisfied."
  },
  {
    "reqid" :"INSTRUMENTATION_UI_T_2",
    "parent_reqid" :"",
    "rationale" :"RFP function 11",
    "fulltext" :"RTS shall always satisfy T_2 = UI_T_2_Display",
    "description" :"ENFORCED: in the interval defined by the entire execution.\nTRIGGER:
        ↪ first point in the interval.\nREQUIRES: for every trigger, RES must hold at all
        ↪ time points between (and including) the trigger and the end of the interval."
  },
  {
    "reqid" :"INSTRUMENTATION_2_SETPOINT_T",
    "parent_reqid" :"",
    "rationale" :"RFP function 8",
    "fulltext" :"RTS shall always satisfy\n if UI_Maintenance_2 then Setpoint_T_2 =
        ↪ UI_Setpoint_T_2\n",
    "description" :"ENFORCED: in the interval defined by the entire execution.\nTRIGGER:
        ↪ first point in the interval.\nREQUIRES: for every trigger, RES must hold at all
        ↪ time points between (and including) the trigger and the end of the interval."
  },
  {
    "reqid" :"INSTRUMENTATION_2_MANUAL_TRIP_T",
    "parent_reqid" :"",
    "rationale" :"RFP function 10",
    "fulltext" :"RTS shall always satisfy\n if UI_Maintenance_2 & UI_Manual_Trip_T_2 then
        ↪ Trip_T_2\n",
    "description" :"ENFORCED: in the interval defined by the entire execution.\nTRIGGER:
        ↪ first point in the interval.\nREQUIRES: for every trigger, RES must hold at all
        ↪ time points between (and including) the trigger and the end of the interval."
  },
  {
    "reqid" :"INSTRUMENTATION_UI_P_2",
    "parent_reqid" :"",
    "rationale" :"RFP function 11",
    "fulltext" :"RTS shall always satisfy P_2 = UI_P_2_Display",
    "description" :"ENFORCED: in the interval defined by the entire execution.\nTRIGGER:
        ↪ first point in the interval.\nREQUIRES: for every trigger, RES must hold at all
        ↪ time points between (and including) the trigger and the end of the interval."
  },
  {
    "reqid" :"INSTRUMENTATION_2_SETPOINT_P",
    "parent_reqid" :"",
    "rationale" :"RFP function 8",
    "fulltext" :"RTS shall always satisfy\n if UI_Maintenance_2 then Setpoint_P_2 =
        ↪ UI_Setpoint_P_2\n",
    "description" :"ENFORCED: in the interval defined by the entire execution.\nTRIGGER:
        ↪ first point in the interval.\nREQUIRES: for every trigger, RES must hold at all
```

```
        ↪ time points between (and including) the trigger and the end of the interval."
582   },
583   {
584     "reqid" :"INSTRUMENTATION_2_MANUAL_TRIP_P",
585     "parent_reqid" :"",
586     "rationale" :"RFP function 10",
587     "fulltext" :"RTS shall always satisfy\n if UI_Maintenance_2 & UI_Manual_Trip_P_2 then
        ↪ Trip_P_2\n",
588     "description" :"ENFORCED: in the interval defined by the entire execution.\nTRIGGER:
        ↪ first point in the interval.\nREQUIRES: for every trigger, RES must hold at all
        ↪ time points between (and including) the trigger and the end of the interval."
589   },
590   {
591     "reqid" :"INSTRUMENTATION_UI_S_2",
592     "parent_reqid" :"",
593     "rationale" :"RFP function 11",
594     "fulltext" :"RTS shall always satisfy S_2 = UI_S_2_Display",
595     "description" :"ENFORCED: in the interval defined by the entire execution.\nTRIGGER:
        ↪ first point in the interval.\nREQUIRES: for every trigger, RES must hold at all
        ↪ time points between (and including) the trigger and the end of the interval."
596   },
597   {
598     "reqid" :"INSTRUMENTATION_2_SETPOINT_S",
599     "parent_reqid" :"",
600     "rationale" :"RFP function 8",
601     "fulltext" :"RTS shall always satisfy\n if UI_Maintenance_2 then Setpoint_S_2 =
        ↪ UI_Setpoint_S_2\n",
602     "description" :"ENFORCED: in the interval defined by the entire execution.\nTRIGGER:
        ↪ first point in the interval.\nREQUIRES: for every trigger, RES must hold at all
        ↪ time points between (and including) the trigger and the end of the interval."
603   },
604   {
605     "reqid" :"INSTRUMENTATION_2_MANUAL_TRIP_S",
606     "parent_reqid" :"",
607     "rationale" :"RFP function 10",
608     "fulltext" :"RTS shall always satisfy\n if UI_Maintenance_2 & UI_Manual_Trip_S_2 then
        ↪ Trip_S_2\n",
609     "description" :"ENFORCED: in the interval defined by the entire execution.\nTRIGGER:
        ↪ first point in the interval.\nREQUIRES: for every trigger, RES must hold at all
        ↪ time points between (and including) the trigger and the end of the interval."
610   },
611   {
612     "reqid" :"INSTRUMENTATION_3_BYPASS_T_Display",
613     "parent_reqid" :"",
614     "rationale" :"RFP function 13",
615     "fulltext" :"RTS shall always satisfy UI_Bypass_T_3_Display = Bypass_T_3\n",
616     "description" :"ENFORCED: in the interval defined by the entire execution.\nTRIGGER:
        ↪ first point in the interval.\nREQUIRES: for every trigger, RES must hold at all
        ↪ time points between (and including) the trigger and the end of the interval."
617   },
618   {
619     "reqid" :"INSTRUMENTATION_3_BYPASS_T",
620     "parent_reqid" :"",
621     "rationale" :"RFP function 9:configure in maintenance mode to bypass an instrument
        ↪ channel (this requirement sets the system value from the UI)\n",
622     "fulltext" :"RTS shall always satisfy\n if UI_Maintenance_3\n then (UI_Bypass_T_3) =
        ↪ Bypass_T_3)\n",
623     "description" :"ENFORCED: in the interval defined by the entire execution.\nTRIGGER:
        ↪ first point in the interval.\nREQUIRES: for every trigger, RES must hold at all
        ↪ time points between (and including) the trigger and the end of the interval."
624   },
625   {
626     "reqid" :"CONST_INSTRUMENTATION_3_NO_BYPASS_T",
627     "parent_reqid" :"INSTRUMENTATION_3_BYPASS_T",
628     "rationale" :"RFP function 9:configure in maintenance mode to bypass an instrument
        ↪ channel (this requirement does not allow bypass to turn on during normal
        ↪ operation)\n",
```

```
    "fulltext" :"When !UI_Maintenance_3 & !Bypass_T_3 RTS shall, until UI_Maintenance_3 &
        ↪ UI_Bypass_T_3, satisfy\n !Bypass_T_3\n",
    "description" :"ENFORCED: in the interval defined by the entire execution.\nTRIGGER:
        ↪ first point in the interval if <b><i>(! UI_Maintenance_3 & ! Bypass_T_3)</i></b>
        ↪  is true and any point in the interval where <b><i>(! UI_Maintenance_3 & !
        ↪ Bypass_T_3)</i></b> becomes true (from false).\nREQUIRES: for every trigger, RES
        ↪  must remain true until (but not necessarily including) the point where the stop
        ↪  condition holds, or to the end of the interval. If the stop condition never
        ↪ occurs, RES must hold until the end of the scope, or forever. If the stop
        ↪ condition holds at the trigger, the requirement is satisfied."
  },
  {
    "reqid" :"CONST_INSTRUMENTATION_3_BYPASS_T",
    "parent_reqid" :"INSTRUMENTATION_3_BYPASS_T",
    "rationale" :"RFP function 9:configure in maintenance mode to bypass an instrument
        ↪ channel (this requirement does not allow bypass to turn off during normal
        ↪ operation)\n",
    "fulltext" :"When !UI_Maintenance_3 & Bypass_T_3 RTS shall, until UI_Maintenance_3 & !
        ↪ UI_Bypass_T_3, satisfy\n Bypass_T_3\n",
    "description" :"ENFORCED: in the interval defined by the entire execution.\nTRIGGER:
        ↪ first point in the interval if <b><i>(! UI_Maintenance_3 & Bypass_T_3)</i></b>
        ↪ is true and any point in the interval where <b><i>(! UI_Maintenance_3 &
        ↪ Bypass_T_3)</i></b> becomes true (from false).\nREQUIRES: for every trigger, RES
        ↪  must remain true until (but not necessarily including) the point where the stop
        ↪  condition holds, or to the end of the interval. If the stop condition never
        ↪ occurs, RES must hold until the end of the scope, or forever. If the stop
        ↪ condition holds at the trigger, the requirement is satisfied."
  },
  {
    "reqid" :"INSTRUMENTATION_3_BYPASS_P_Display",
    "parent_reqid" :"",
    "rationale" :"RFP function 13",
    "fulltext" :"RTS shall always satisfy UI_Bypass_P_3_Display = Bypass_P_3\n",
    "description" :"ENFORCED: in the interval defined by the entire execution.\nTRIGGER:
        ↪ first point in the interval.\nREQUIRES: for every trigger, RES must hold at all
        ↪ time points between (and including) the trigger and the end of the interval."
  },
  {
    "reqid" :"INSTRUMENTATION_3_BYPASS_P",
    "parent_reqid" :"",
    "rationale" :"RFP function 9:configure in maintenance mode to bypass an instrument
        ↪ channel (this requirement sets the system value from the UI)\n",
    "fulltext" :"RTS shall always satisfy\n if UI_Maintenance_3\n then (UI_Bypass_P_3) =
        ↪ Bypass_P_3)\n",
    "description" :"ENFORCED: in the interval defined by the entire execution.\nTRIGGER:
        ↪ first point in the interval.\nREQUIRES: for every trigger, RES must hold at all
        ↪ time points between (and including) the trigger and the end of the interval."
  },
  {
    "reqid" :"CONST_INSTRUMENTATION_3_NO_BYPASS_P",
    "parent_reqid" :"INSTRUMENTATION_3_BYPASS_P",
    "rationale" :"RFP function 9:configure in maintenance mode to bypass an instrument
        ↪ channel (this requirement does not allow bypass to turn on during normal
        ↪ operation)\n",
    "fulltext" :"When !UI_Maintenance_3 & !Bypass_P_3 RTS shall, until UI_Maintenance_3 &
        ↪ UI_Bypass_P_3, satisfy\n !Bypass_P_3\n",
    "description" :"ENFORCED: in the interval defined by the entire execution.\nTRIGGER:
        ↪ first point in the interval if <b><i>(! UI_Maintenance_3 & ! Bypass_P_3)</i></b>
        ↪  is true and any point in the interval where <b><i>(! UI_Maintenance_3 & !
        ↪ Bypass_P_3)</i></b> becomes true (from false).\nREQUIRES: for every trigger, RES
        ↪  must remain true until (but not necessarily including) the point where the stop
        ↪  condition holds, or to the end of the interval. If the stop condition never
        ↪ occurs, RES must hold until the end of the scope, or forever. If the stop
        ↪ condition holds at the trigger, the requirement is satisfied."
  },
  {
    "reqid" :"CONST_INSTRUMENTATION_3_BYPASS_P",
    "parent_reqid" :"INSTRUMENTATION_3_BYPASS_P",
```

```
663     "rationale" :"RFP function 9:configure in maintenance mode to bypass an instrument
            ↪ channel (this requirement does not allow bypass to turn off during normal
            ↪ operation)\n",
664     "fulltext" :"When !UI_Maintenance_3 & Bypass_P_3 RTS shall, until UI_Maintenance_3 & !
            ↪ UI_Bypass_P_3, satisfy\n Bypass_P_3\n",
665     "description" :"ENFORCED: in the interval defined by the entire execution.\nTRIGGER:
            ↪ first point in the interval if <b><i>(! UI_Maintenance_3 & Bypass_P_3)</i></b>
            ↪ is true and any point in the interval where <b><i>(! UI_Maintenance_3 &
            ↪ Bypass_P_3)</i></b> becomes true (from false).\nREQUIRES: for every trigger, RES
            ↪  must remain true until (but not necessarily including) the point where the stop
            ↪  condition holds, or to the end of the interval. If the stop condition never
            ↪ occurs, RES must hold until the end of the scope, or forever. If the stop
            ↪ condition holds at the trigger, the requirement is satisfied."
666   },
667   {
668     "reqid" :"INSTRUMENTATION_UI_T_3",
669     "parent_reqid" :"",
670     "rationale" :"RFP function 11",
671     "fulltext" :"RTS shall always satisfy T_3 = UI_T_3_Display",
672     "description" :"ENFORCED: in the interval defined by the entire execution.\nTRIGGER:
            ↪ first point in the interval.\nREQUIRES: for every trigger, RES must hold at all
            ↪ time points between (and including) the trigger and the end of the interval."
673   },
674   {
675     "reqid" :"INSTRUMENTATION_3_SETPOINT_T",
676     "parent_reqid" :"",
677     "rationale" :"RFP function 8",
678     "fulltext" :"RTS shall always satisfy\n if UI_Maintenance_3 then Setpoint_T_3 =
            ↪ UI_Setpoint_T_3\n",
679     "description" :"ENFORCED: in the interval defined by the entire execution.\nTRIGGER:
            ↪ first point in the interval.\nREQUIRES: for every trigger, RES must hold at all
            ↪ time points between (and including) the trigger and the end of the interval."
680   },
681   {
682     "reqid" :"INSTRUMENTATION_3_MANUAL_TRIP_T",
683     "parent_reqid" :"",
684     "rationale" :"RFP function 10",
685     "fulltext" :"RTS shall always satisfy\n if UI_Maintenance_3 & UI_Manual_Trip_T_3 then
            ↪ Trip_T_3\n",
686     "description" :"ENFORCED: in the interval defined by the entire execution.\nTRIGGER:
            ↪ first point in the interval.\nREQUIRES: for every trigger, RES must hold at all
            ↪ time points between (and including) the trigger and the end of the interval."
687   },
688   {
689     "reqid" :"INSTRUMENTATION_UI_P_3",
690     "parent_reqid" :"",
691     "rationale" :"RFP function 11",
692     "fulltext" :"RTS shall always satisfy P_3 = UI_P_3_Display",
693     "description" :"ENFORCED: in the interval defined by the entire execution.\nTRIGGER:
            ↪ first point in the interval.\nREQUIRES: for every trigger, RES must hold at all
            ↪ time points between (and including) the trigger and the end of the interval."
694   },
695   {
696     "reqid" :"INSTRUMENTATION_3_SETPOINT_P",
697     "parent_reqid" :"",
698     "rationale" :"RFP function 8",
699     "fulltext" :"RTS shall always satisfy\n if UI_Maintenance_3 then Setpoint_P_3 =
            ↪ UI_Setpoint_P_3\n",
700     "description" :"ENFORCED: in the interval defined by the entire execution.\nTRIGGER:
            ↪ first point in the interval.\nREQUIRES: for every trigger, RES must hold at all
            ↪ time points between (and including) the trigger and the end of the interval."
701   },
702   {
703     "reqid" :"INSTRUMENTATION_3_MANUAL_TRIP_P",
704     "parent_reqid" :"",
705     "rationale" :"RFP function 10",
706     "fulltext" :"RTS shall always satisfy\n if UI_Maintenance_3 & UI_Manual_Trip_P_3 then
            ↪ Trip_P_3\n",
```

```
    "description" :"ENFORCED: in the interval defined by the entire execution.\nTRIGGER:
        ↪ first point in the interval.\nREQUIRES: for every trigger, RES must hold at all
        ↪ time points between (and including) the trigger and the end of the interval."
  },
  {
    "reqid" :"INSTRUMENTATION_UI_S_3",
    "parent_reqid" :"",
    "rationale" :"RFP function 11",
    "fulltext" :"RTS shall always satisfy S_3 = UI_S_3_Display",
    "description" :"ENFORCED: in the interval defined by the entire execution.\nTRIGGER:
        ↪ first point in the interval.\nREQUIRES: for every trigger, RES must hold at all
        ↪ time points between (and including) the trigger and the end of the interval."
  },
  {
    "reqid" :"INSTRUMENTATION_3_SETPOINT_S",
    "parent_reqid" :"",
    "rationale" :"RFP function 8",
    "fulltext" :"RTS shall always satisfy\n if UI_Maintenance_3 then Setpoint_S_3 =
        ↪ UI_Setpoint_S_3\n",
    "description" :"ENFORCED: in the interval defined by the entire execution.\nTRIGGER:
        ↪ first point in the interval.\nREQUIRES: for every trigger, RES must hold at all
        ↪ time points between (and including) the trigger and the end of the interval."
  },
  {
    "reqid" :"INSTRUMENTATION_3_MANUAL_TRIP_S",
    "parent_reqid" :"",
    "rationale" :"RFP function 10",
    "fulltext" :"RTS shall always satisfy\n if UI_Maintenance_3 & UI_Manual_Trip_S_3 then
        ↪ Trip_S_3\n",
    "description" :"ENFORCED: in the interval defined by the entire execution.\nTRIGGER:
        ↪ first point in the interval.\nREQUIRES: for every trigger, RES must hold at all
        ↪ time points between (and including) the trigger and the end of the interval."
  }
]
```

Listing C.2: Requirements from FRET.

```
[
  {
    "reqid" :"INSTRUMENTATION_SET_MANUAL_TRIP_TEMPERATURE",
    "parent_reqid" :"INSTRUMENTATION_TRIP_TEMPERATURE",
    "rationale" :"RFP [10]",
    "fulltext" :"Upon MAINTENANCE & TEMPERATURE_MODE = 2Instrumentation shall, until
        ↪ MAINTENANCE & !(TEMPERATURE_MODE = 2), satisfy TRIP_TEMPERATURE",
    "description" :"ENFORCED: in the interval defined by the entire execution.\nTRIGGER:
        ↪ first point in the interval if <b><i>(MAINTENANCE & TEMPERATURE_MODE = 2)</i></b
        ↪ > is true and any point in the interval where <b><i>(MAINTENANCE &
        ↪ TEMPERATURE_MODE = 2)</i></b> becomes true (from false).\nREQUIRES: for every
        ↪ trigger, RES must remain true until (but not necessarily including) the point
        ↪ where the stop condition holds, or to the end of the interval. If the stop
        ↪ condition never occurs, RES must hold until the end of the scope, or forever. If
        ↪  the stop condition holds at the trigger, the requirement is satisfied."
  },
  {
    "reqid" :"INSTRUMENTATION_SENSOR_TRIP_PRESSURE",
    "parent_reqid" :"INSTRUMENTATION_TRIP_PRESSURE",
    "rationale" :"RFP [1]",
    "fulltext" :"Upon MAINTENANCE & PRESSURE_MODE = 1Instrumentation shall, until MAINTENANCE
        ↪  & !(PRESSURE_MODE = 1), satisfy (if SENSOR_PRESSURE > SETPOINT_PRESSURE then
        ↪ TRIP_PRESSURE) & (if TRIP_PRESSURE then SENSOR_PRESSURE > SETPOINT_PRESSURE)",
    "description" :"ENFORCED: in the interval defined by the entire execution.\nTRIGGER:
        ↪ first point in the interval if <b><i>(MAINTENANCE & PRESSURE_MODE = 1)</i></b>
        ↪ is true and any point in the interval where <b><i>(MAINTENANCE & PRESSURE_MODE =
        ↪  1)</i></b> becomes true (from false).\nREQUIRES: for every trigger, RES must
        ↪ remain true until (but not necessarily including) the point where the stop
        ↪ condition holds, or to the end of the interval. If the stop condition never
        ↪ occurs, RES must hold until the end of the scope, or forever. If the stop
```

```
        ↪ condition holds at the trigger, the requirement is satisfied."
  },
  {
    "reqid" :"INSTRUMENTATION_SET_SETPOINT_PRESSURE",
    "parent_reqid" :"INSTRUMENTATION_TRIP_PRESSURE",
    "rationale" :"RFP [8]",
    "fulltext" :"Upon (MAINTENANCE & SET_SETPOINT_PRESSURE) Instrumentation shall, until
        ↪ MAINTENANCE & SET_SETPOINT_PRESSURE, satisfy SETPOINT_PRESSURE =
        ↪ INPUT_SETPOINT_PRESSURE",
    "description" :"ENFORCED: in the interval defined by the entire execution.\nTRIGGER:
        ↪ first point in the interval if <b><i>(( MAINTENANCE & SET_SETPOINT_PRESSURE ))</
        ↪ i></b> is true and any point in the interval where <b><i>(( MAINTENANCE &
        ↪ SET_SETPOINT_PRESSURE ))</i></b> becomes true (from false).\nREQUIRES: for every
        ↪  trigger, RES must remain true until (but not necessarily including) the point
        ↪ where the stop condition holds, or to the end of the interval. If the stop
        ↪ condition never occurs, RES must hold until the end of the scope, or forever. If
        ↪  the stop condition holds at the trigger, the requirement is satisfied."
  },
  {
    "reqid" :"INSTRUMENTATION_SENSOR_TRIP_SATURATION",
    "parent_reqid" :"INSTRUMENTATION_TRIP_SATURATION",
    "rationale" :"RFP [3]",
    "fulltext" :"Upon MAINTENANCE & SATURATION_MODE = 1Instrumentation shall, until
        ↪ MAINTENANCE & !(SATURATION_MODE = 1) satisfy (if SATURATION_FUNCTION_VALUE <
        ↪ SETPOINT_SATURATION then TRIP_SATURATION) & (if TRIP_SATURATION then
        ↪ SATURATION_FUNCTION_VALUE < SETPOINT_SATURATION)",
    "description" :"ENFORCED: in the interval defined by the entire execution.\nTRIGGER:
        ↪ first point in the interval if <b><i>(MAINTENANCE & SATURATION_MODE = 1)</i></b>
        ↪  is true and any point in the interval where <b><i>(MAINTENANCE &
        ↪ SATURATION_MODE = 1)</i></b> becomes true (from false).\nREQUIRES: for every
        ↪ trigger, RES must remain true until (but not necessarily including) the point
        ↪ where the stop condition holds, or to the end of the interval. If the stop
        ↪ condition never occurs, RES must hold until the end of the scope, or forever. If
        ↪  the stop condition holds at the trigger, the requirement is satisfied."
  },
  {
    "reqid" :"INSTRUMENTATION_SET_MANUAL_TRIP_PRESSURE",
    "parent_reqid" :"INSTRUMENTATION_TRIP_PRESSURE",
    "rationale" :"RFP [10]",
    "fulltext" :"Upon MAINTENANCE & PRESSURE_MODE = 2Instrumentation shall, until MAINTENANCE
        ↪  & !(PRESSURE_MODE = 2), satisfy TRIP_PRESSURE",
    "description" :"ENFORCED: in the interval defined by the entire execution.\nTRIGGER:
        ↪ first point in the interval if <b><i>(MAINTENANCE & PRESSURE_MODE = 2)</i></b>
        ↪ is true and any point in the interval where <b><i>(MAINTENANCE & PRESSURE_MODE =
        ↪  2)</i></b> becomes true (from false).\nREQUIRES: for every trigger, RES must
        ↪ remain true until (but not necessarily including) the point where the stop
        ↪ condition holds, or to the end of the interval. If the stop condition never
        ↪ occurs, RES must hold until the end of the scope, or forever. If the stop
        ↪ condition holds at the trigger, the requirement is satisfied."
  },
  {
    "reqid" :"INSTRUMENTATION_SET_BYPASS_SATURATION",
    "parent_reqid" :"INSTRUMENTATION_TRIP_SATURATION",
    "rationale" :"",
    "fulltext" :"Upon MAINTENANCE & SATURATION_MODE = 0Instrumentation shall, until
        ↪ MAINTENANCE & !(SATURATION_MODE = 0), satisfy !TRIP_SATURATION",
    "description" :"ENFORCED: in the interval defined by the entire execution.\nTRIGGER:
        ↪ first point in the interval if <b><i>(MAINTENANCE & SATURATION_MODE = 0)</i></b>
        ↪  is true and any point in the interval where <b><i>(MAINTENANCE &
        ↪ SATURATION_MODE = 0)</i></b> becomes true (from false).\nREQUIRES: for every
        ↪ trigger, RES must remain true until (but not necessarily including) the point
        ↪ where the stop condition holds, or to the end of the interval. If the stop
        ↪ condition never occurs, RES must hold until the end of the scope, or forever. If
        ↪  the stop condition holds at the trigger, the requirement is satisfied."
  },
  {
    "reqid" :"ACTUATION_LOGIC_DEVICE_0",
    "parent_reqid" :"",
```

```
47      "rationale" :"RFP [5,6]",
48      "fulltext" :"Actuation_Logic shall always satisfy ((VOTE_ACTUATE_DEVICE_0 |
            ↪ MANUAL_ACTUATE_DEVICE_0) => ACTUATE_DEVICE_0) & (ACTUATE_DEVICE_0 => (
            ↪ VOTE_ACTUATE_DEVICE_0 | MANUAL_ACTUATE_DEVICE_0))",
49      "description" :"ENFORCED: in the interval defined by the entire execution.\nTRIGGER:
            ↪ first point in the interval.\nREQUIRES: for every trigger, RES must hold at all
            ↪ time points between (and including) the trigger and the end of the interval."
50    },
51    {
52      "reqid" :"INSTRUMENTATION_SET_SETPOINT_SATURATION",
53      "parent_reqid" :"INSTRUMENTATION_TRIP_SATURATION",
54      "rationale" :"RFP [8]",
55      "fulltext" :"Upon (MAINTENANCE & SET_SETPOINT_SATURATION) Instrumentation shall, until
            ↪ MAINTENANCE & SET_SETPOINT_SATURATION, satisfy SETPOINT_SATURATION =
            ↪ INPUT_SETPOINT_SATURATION",
56      "description" :"ENFORCED: in the interval defined by the entire execution.\nTRIGGER:
            ↪ first point in the interval if <b><i>(( MAINTENANCE & SET_SETPOINT_SATURATION ))
            ↪ </i></b> is true and any point in the interval where <b><i>(( MAINTENANCE &
            ↪ SET_SETPOINT_SATURATION ))</i></b> becomes true (from false).\nREQUIRES: for
            ↪ every trigger, RES must remain true until (but not necessarily including) the
            ↪ point where the stop condition holds, or to the end of the interval. If the stop
            ↪  condition never occurs, RES must hold until the end of the scope, or forever.
            ↪ If the stop condition holds at the trigger, the requirement is satisfied."
57    },
58    {
59      "reqid" :"INSTRUMENTATION_SET_BYPASS_PRESSURE",
60      "parent_reqid" :"INSTRUMENTATION_TRIP_PRESSURE",
61      "rationale" :"RFP [9]",
62      "fulltext" :"Upon MAINTENANCE & PRESSURE_MODE = 0Instrumentation shall, until MAINTENANCE
            ↪  & !(PRESSURE_MODE = 0), satisfy !TRIP_PRESSURE",
63      "description" :"ENFORCED: in the interval defined by the entire execution.\nTRIGGER:
            ↪ first point in the interval if <b><i>(MAINTENANCE & PRESSURE_MODE = 0)</i></b>
            ↪ is true and any point in the interval where <b><i>(MAINTENANCE & PRESSURE_MODE =
            ↪  0)</i></b> becomes true (from false).\nREQUIRES: for every trigger, RES must
            ↪ remain true until (but not necessarily including) the point where the stop
            ↪ condition holds, or to the end of the interval. If the stop condition never
            ↪ occurs, RES must hold until the end of the scope, or forever. If the stop
            ↪ condition holds at the trigger, the requirement is satisfied."
64    },
65    {
66      "reqid" :"INSTRUMENTATION_SET_MANUAL_TRIP_SATURATION",
67      "parent_reqid" :"INSTRUMENTATION_TRIP_SATURATION",
68      "rationale" :"RFP [10]",
69      "fulltext" :"Upon MAINTENANCE & SATURATION_MODE = 2Instrumentation shall, until
            ↪ MAINTENANCE & !(SATURATION_MODE = 2), satisfy TRIP_SATURATION",
70      "description" :"ENFORCED: in the interval defined by the entire execution.\nTRIGGER:
            ↪ first point in the interval if <b><i>(MAINTENANCE & SATURATION_MODE = 2)</i></b>
            ↪  is true and any point in the interval where <b><i>(MAINTENANCE &
            ↪ SATURATION_MODE = 2)</i></b> becomes true (from false).\nREQUIRES: for every
            ↪ trigger, RES must remain true until (but not necessarily including) the point
            ↪ where the stop condition holds, or to the end of the interval. If the stop
            ↪ condition never occurs, RES must hold until the end of the scope, or forever. If
            ↪  the stop condition holds at the trigger, the requirement is satisfied."
71    },
72    {
73      "reqid" :"ACTUATION_LOGIC_VOTE_TEMPERATURE",
74      "parent_reqid" :"ACTUATION_LOGIC_VOTE_DEVICE_0",
75      "rationale" :"RFP Actuation Logic Architecture, [2,4]",
76      "fulltext" :"Actuation_Logic shall always satisfy (((TRIP_TEMPERATURE_0 &
            ↪ TRIP_TEMPERATURE_1) | ((TRIP_TEMPERATURE_0 | TRIP_TEMPERATURE_1) & (
            ↪ TRIP_TEMPERATURE_2 | TRIP_TEMPERATURE_3)) | (TRIP_TEMPERATURE_2 &
            ↪ TRIP_TEMPERATURE_3)) => VOTE_TRIP_TEMPERATURE) & (VOTE_TRIP_TEMPERATURE => ((
            ↪ TRIP_TEMPERATURE_0 & TRIP_TEMPERATURE_1) | ((TRIP_TEMPERATURE_0 |
            ↪ TRIP_TEMPERATURE_1) & (TRIP_TEMPERATURE_2 | TRIP_TEMPERATURE_3)) | (
            ↪ TRIP_TEMPERATURE_2 & TRIP_TEMPERATURE_3)))",
77      "description" :"ENFORCED: in the interval defined by the entire execution.\nTRIGGER:
            ↪ first point in the interval.\nREQUIRES: for every trigger, RES must hold at all
            ↪ time points between (and including) the trigger and the end of the interval."
```

```
78    },
79    {
80      "reqid" :"ACTUATION_LOGIC_DEVICE_1",
81      "parent_reqid" :"",
82      "rationale" :"RFP [5,6]",
83      "fulltext" :"Actuation_Logic shall always satisfy (VOTE_ACTUATE_DEVICE_1 |
            ↪ MANUAL_ACTUATE_DEVICE_1 => ACTUATE_DEVICE_1) & (ACTUATE_DEVICE_1 =>
            ↪ VOTE_ACTUATE_DEVICE_1 | MANUAL_ACTUATE_DEVICE_1)",
84      "description" :"ENFORCED: in the interval defined by the entire execution.\nTRIGGER:
            ↪ first point in the interval.\nREQUIRES: for every trigger, RES must hold at all
            ↪ time points between (and including) the trigger and the end of the interval."
85    },
86    {
87      "reqid" :"ACTUATION_LOGIC_VOTE_PRESSURE",
88      "parent_reqid" :"ACTUATION_LOGIC_VOTE_DEVICE_0",
89      "rationale" :"RFP Actuation Logic Architecture, [1,4]",
90      "fulltext" :"Actuation_Logic shall always satisfy (IF ((TRIP_PRESSURE_0 & TRIP_PRESSURE_1
            ↪ ) | (( TRIP_PRESSURE_0 | TRIP_PRESSURE_1) & (TRIP_PRESSURE_2 | TRIP_PRESSURE_3))
            ↪  | (TRIP_PRESSURE_2 & TRIP_PRESSURE_3)) THEN VOTE_TRIP_PRESSURE) & (IF
            ↪ VOTE_TRIP_PRESSURE THEN ((TRIP_PRESSURE_0 & TRIP_PRESSURE_1) | (( TRIP_PRESSURE_
            ↪ 0 | TRIP_PRESSURE_1) & (TRIP_PRESSURE_2 | TRIP_PRESSURE_3)) | (TRIP_PRESSURE_2 &
            ↪  TRIP_PRESSURE_3)))",
91      "description" :"ENFORCED: in the interval defined by the entire execution.\nTRIGGER:
            ↪ first point in the interval.\nREQUIRES: for every trigger, RES must hold at all
            ↪ time points between (and including) the trigger and the end of the interval."
92    },
93    {
94      "reqid" :"INSTRUMENTATION_TRIP_TEMPERATURE",
95      "parent_reqid" :"",
96      "rationale" :"RFP [3,10]",
97      "fulltext" :"Instrumentation shall always satisfy true",
98      "description" :"ENFORCED: in the interval defined by the entire execution.\nTRIGGER:
            ↪ first point in the interval.\nREQUIRES: for every trigger, RES must hold at all
            ↪ time points between (and including) the trigger and the end of the interval."
99    },
100   {
101     "reqid" :"INSTRUMENTATION_RESET",
102     "parent_reqid" :"",
103     "rationale" :"",
104     "fulltext" :"AFTER RESET Instrumentation shall immediately satisfy (MAINTENANCE &
            ↪ PRESSURE_TRIP_MODE = 0& TEMPERATURE_TRIP_MODE = 0& SATURATION_TRIP_MODE = 0)",
105     "description" :"ENFORCED: in the interval (if defined) starting strictly after the first
            ↪ <b><i>RESET</i></b> interval and spanning to the end of the execution.\nTRIGGER:
            ↪  first point in the interval.\nREQUIRES: for every trigger, if trigger holds
            ↪ then RES also holds at the same time point."
106   },
107   {
108     "reqid" :"INSTRUMENTATION_TRIP_SATURATION",
109     "parent_reqid" :"",
110     "rationale" :"RFP [2,10]",
111     "fulltext" :"Instrumentation shall always satisfy true",
112     "description" :"ENFORCED: in the interval defined by the entire execution.\nTRIGGER:
            ↪ first point in the interval.\nREQUIRES: for every trigger, RES must hold at all
            ↪ time points between (and including) the trigger and the end of the interval."
113   },
114   {
115     "reqid" :"CORE_UI_INSTRUMENTATION",
116     "parent_reqid" :"",
117     "rationale" :"RFP [11,12,13]",
118     "fulltext" :"Core shall always satisfy forAll_instr_i & INSTRUMENTATION_i_PRESSURE =
            ↪ UI_i_PRESSURE & INSTRUMENTATION_i_TEMPERATURE = UI_i_TEMPERATURE &
            ↪ INSTRUMENTATION_i_SATURATION = UI_i_SATURATION &
            ↪ INSTRUMENTATION_i_BYPASS_PRESSURE = UI_i_BYPASS_PRESSURE &
            ↪ INSTRUMENTATION_i_BYPASS_TEMPERATURE = UI_i_BYPASS_TEMPERATURE &
            ↪ INSTRUMENTATION_i_TRIP_PRESSURE = UI_i_TRIP_PRESSURE &
            ↪ INSTRUMENTATION_i_TRIP_TEMPERATURE = UI_i_TRIP_TEMPERATURE &
            ↪ INSTRUMENTATION_i_TRIP_SATURATION = UI_i_TRIP_SATURATION",
119     "description" :"ENFORCED: in the interval defined by the entire execution.\nTRIGGER:
```

```
        ↪ first point in the interval.\nREQUIRES: for every trigger, RES must hold at all
        ↪ time points between (and including) the trigger and the end of the interval."
  },
  {
    "reqid" :"SELF_TEST",
    "parent_reqid" :"",
    "rationale" :"",
    "fulltext" :"Upon TEST_FAIL Core shall always satisfy OUTPUT_TEST_FAIL",
    "description" :"ENFORCED: in the interval defined by the entire execution.\nTRIGGER:
        ↪ first point in the interval if <b><i>(TEST_FAIL)</i></b> is true and any point
        ↪ in the interval where <b><i>(TEST_FAIL)</i></b> becomes true (from false).\
        ↪ nREQUIRES: for every trigger, RES must hold at all time points between (and
        ↪ including) the trigger and the end of the interval."
  },
  {
    "reqid" :"INSTRUMENTATION_SET_MAINTENANCE",
    "parent_reqid" :"",
    "rationale" :"RFP [7]",
    "fulltext" :"Upon SET_MAINTENANCE Instrumentation shall, until UNSET_MAINTENANCE, satisfy
        ↪  MAINTENANCE",
    "description" :"ENFORCED: in the interval defined by the entire execution.\nTRIGGER:
        ↪ first point in the interval if <b><i>(SET_MAINTENANCE)</i></b> is true and any
        ↪ point in the interval where <b><i>(SET_MAINTENANCE)</i></b> becomes true (from
        ↪ false).\nREQUIRES: for every trigger, RES must remain true until (but not
        ↪ necessarily including) the point where the stop condition holds, or to the end
        ↪ of the interval. If the stop condition never occurs, RES must hold until the end
        ↪  of the scope, or forever. If the stop condition holds at the trigger, the
        ↪ requirement is satisfied."
  },
  {
    "reqid" :"ACTUATION_LOGIC_MANUAL_DEVICE_0",
    "parent_reqid" :"ACTUATION_LOGIC_DEVICE_0",
    "rationale" :"RFP 6",
    "fulltext" :"Upon SET_MANUAL_ACTUATE_DEVICE_0 Actuation_Logic shall, until
        ↪ UNSET_MANUAL_ACTUATE_DEVICE_0, satisfy MANUAL_ACTUATE_DEVICE_0",
    "description" :"ENFORCED: in the interval defined by the entire execution.\nTRIGGER:
        ↪ first point in the interval if <b><i>(SET_MANUAL_ACTUATE_DEVICE_0)</i></b> is
        ↪ true and any point in the interval where <b><i>(SET_MANUAL_ACTUATE_DEVICE_0)</i
        ↪ ></b> becomes true (from false).\nREQUIRES: for every trigger, RES must remain
        ↪ true until (but not necessarily including) the point where the stop condition
        ↪ holds, or to the end of the interval. If the stop condition never occurs, RES
        ↪ must hold until the end of the scope, or forever. If the stop condition holds at
        ↪  the trigger, the requirement is satisfied."
  },
  {
    "reqid" :"ACTUATION_LOGIC_VOTE_SATURATION",
    "parent_reqid" :"ACTUATION_LOGIC_VOTE_DEVICE_1",
    "rationale" :"RFP Actuation Logic Architecture, [3,4]",
    "fulltext" :"Actuation_Logic shall always satisfy (((TRIP_SATURATION_0 & TRIP_SATURATION_
        ↪ 1) | (( TRIP_SATURATION_0 | TRIP_SATURATION_1) & (TRIP_SATURATION_2 |
        ↪ TRIP_SATURATION_3)) | (TRIP_SATURATION_2 & TRIP_SATURATION_3)) =>
        ↪ VOTE_TRIP_SATURATION) & (VOTE_TRIP_SATURATION => ((TRIP_SATURATION_0 &
        ↪ TRIP_SATURATION_1) | (( TRIP_SATURATION_0 | TRIP_SATURATION_1) & (
        ↪ TRIP_SATURATION_2 | TRIP_SATURATION_3)) | (TRIP_SATURATION_2 & TRIP_SATURATION_3
        ↪ )))",
    "description" :"ENFORCED: in the interval defined by the entire execution.\nTRIGGER:
        ↪ first point in the interval.\nREQUIRES: for every trigger, RES must hold at all
        ↪ time points between (and including) the trigger and the end of the interval."
  },
  {
    "reqid" :"ACTUATION_LOGIC_MANUAL_DEVICE_1",
    "parent_reqid" :"ACTUATION_LOGIC_DEVICE_1",
    "rationale" :"RFP 6",
    "fulltext" :"Upon SET_MANUAL_ACTUATE_DEVICE_1 Actuation_Logic shall, until
        ↪ UNSET_MANUAL_ACTUATE_DEVICE_1, satisfy MANUAL_ACTUATE_DEVICE_1",
    "description" :"ENFORCED: in the interval defined by the entire execution.\nTRIGGER:
        ↪ first point in the interval if <b><i>(SET_MANUAL_ACTUATE_DEVICE_1)</i></b> is
        ↪ true and any point in the interval where <b><i>(SET_MANUAL_ACTUATE_DEVICE_1)</i
```

```
        ↪ ></b> becomes true (from false).\nREQUIRES: for every trigger, RES must remain
        ↪ true until (but not necessarily including) the point where the stop condition
        ↪ holds, or to the end of the interval. If the stop condition never occurs, RES
        ↪ must hold until the end of the scope, or forever. If the stop condition holds at
        ↪  the trigger, the requirement is satisfied."
  },
  {
    "reqid" :"ACTUATION_LOGIC_VOTE_DEVICE_1",
    "parent_reqid" :"ACTUATION_LOGIC_DEVICE_1",
    "rationale" :"RFP 5",
    "fulltext" :"Upon VOTE_TRIP_SATURATION Actuation_Logic shall always satisfy
        ↪ VOTE_ACTUATE_DEVICE_1",
    "description" :"ENFORCED: in the interval defined by the entire execution.\nTRIGGER:
        ↪ first point in the interval if <b><i>(VOTE_TRIP_SATURATION)</i></b> is true and
        ↪ any point in the interval where <b><i>(VOTE_TRIP_SATURATION)</i></b> becomes
        ↪ true (from false).\nREQUIRES: for every trigger, RES must hold at all time
        ↪ points between (and including) the trigger and the end of the interval."
  },
  {
    "reqid" :"INSTRUMENTATION_SET_BYPASS_TEMPERATURE",
    "parent_reqid" :"INSTRUMENTATION_TRIP_TEMPERATURE",
    "rationale" :"RFP [9]",
    "fulltext" :"Upon MAINTENANCE & TEMPERATURE_MODE = 0Instrumentation shall, until
        ↪ MAINTENANCE & !(TEMPERATURE_MODE = 0), satisfy !TRIP_TEMPERATURE",
    "description" :"ENFORCED: in the interval defined by the entire execution.\nTRIGGER:
        ↪ first point in the interval if <b><i>(MAINTENANCE & TEMPERATURE_MODE = 0)</i></b
        ↪ > is true and any point in the interval where <b><i>(MAINTENANCE &
        ↪ TEMPERATURE_MODE = 0)</i></b> becomes true (from false).\nREQUIRES: for every
        ↪ trigger, RES must remain true until (but not necessarily including) the point
        ↪ where the stop condition holds, or to the end of the interval. If the stop
        ↪ condition never occurs, RES must hold until the end of the scope, or forever. If
        ↪  the stop condition holds at the trigger, the requirement is satisfied."
  },
  {
    "reqid" :"INSTRUMENTATION_TRIP_PRESSURE",
    "parent_reqid" :"",
    "rationale" :"RFP [1,10]",
    "fulltext" :"Instrumentation shall always satisfy true",
    "description" :"ENFORCED: in the interval defined by the entire execution.\nTRIGGER:
        ↪ first point in the interval.\nREQUIRES: for every trigger, RES must hold at all
        ↪ time points between (and including) the trigger and the end of the interval."
  },
  {
    "reqid" :"ACTUATION_LOGIC_VOTE_DEVICE_0",
    "parent_reqid" :"ACTUATION_LOGIC_DEVICE_0",
    "rationale" :"RFP 5",
    "fulltext" :"Upon VOTE_TRIP_TEMPERATURE | VOTE_TRIP_SATURATION Actuation_Logic shall
        ↪ always satisfy VOTE_ACTUATE_DEVICE_0",
    "description" :"ENFORCED: in the interval defined by the entire execution.\nTRIGGER:
        ↪ first point in the interval if <b><i>(VOTE_TRIP_TEMPERATURE |
        ↪ VOTE_TRIP_SATURATION)</i></b> is true and any point in the interval where <b><i
        ↪ >(VOTE_TRIP_TEMPERATURE | VOTE_TRIP_SATURATION)</i></b> becomes true (from false
        ↪ ).\nREQUIRES: for every trigger, RES must hold at all time points between (and
        ↪ including) the trigger and the end of the interval."
  },
  {
    "reqid" :"INSTRUMENTATION_SENSOR_TRIP_TEMPERATURE",
    "parent_reqid" :"INSTRUMENTATION_TRIP_TEMPERATURE",
    "rationale" :"RFP [2]",
    "fulltext" :"Upon MAINTENANCE & TEMPERATURE_MODE = 1Instrumentation shall, until
        ↪ MAINTENANCE & !(TEMPERATURE_MODE = 1), satisfy (if SENSOR_TEMPERATURE >
        ↪ SETPOINT_TEMPERATURE then TRIP_TEMPERATURE) & (if TRIP_TEMPERATURE then
        ↪ SENSOR_TEMPERATURE > SETPOINT_TEMPERATURE)",
    "description" :"ENFORCED: in the interval defined by the entire execution.\nTRIGGER:
        ↪ first point in the interval if <b><i>(MAINTENANCE & TEMPERATURE_MODE = 1)</i></b
        ↪ > is true and any point in the interval where <b><i>(MAINTENANCE &
        ↪ TEMPERATURE_MODE = 1)</i></b> becomes true (from false).\nREQUIRES: for every
        ↪ trigger, RES must remain true until (but not necessarily including) the point
```

```
        ↪ where the stop condition holds, or to the end of the interval. If the stop
        ↪ condition never occurs, RES must hold until the end of the scope, or forever. If
        ↪  the stop condition holds at the trigger, the requirement is satisfied."
  },
  {
    "reqid" :"INSTRUMENTATION_SET_SETPOINT_TEMPERATURE",
    "parent_reqid" :"INSTRUMENTATION_TRIP_TEMPERATURE",
    "rationale" :"RFP [8]",
    "fulltext" :"Upon (MAINTENANCE & SET_SETPOINT_TEMPERATURE) Instrumentation shall, until
        ↪ MAINTENANCE & SET_SETPOINT_TEMPERATURE, satisfy SETPOINT_TEMPERATURE =
        ↪ INPUT_SETPOINT_TEMPERATURE",
    "description" :"ENFORCED: in the interval defined by the entire execution.\nTRIGGER:
        ↪ first point in the interval if <b><i>(( MAINTENANCE & SET_SETPOINT_TEMPERATURE )
        ↪ )</i></b> is true and any point in the interval where <b><i>(( MAINTENANCE &
        ↪ SET_SETPOINT_TEMPERATURE ))</i></b> becomes true (from false).\nREQUIRES: for
        ↪ every trigger, RES must remain true until (but not necessarily including) the
        ↪ point where the stop condition holds, or to the end of the interval. If the stop
        ↪  condition never occurs, RES must hold until the end of the scope, or forever.
        ↪ If the stop condition holds at the trigger, the requirement is satisfied."
  }
]
```

Appendix D

SysMLv2 Model

D.1 Top-level SysMLv2 Architecture Specification

Listing D.1: Listing SysML Model of HARDENS.

```
/*
 # Reactor Trip System (RTS) High,assurance Demonstrator
 ## project: High Assurance Rigorous Digital Engineering for Nuclear Safety (HARDENS)
 ### copyright (C) 2021 Galois
 ### author: Joe Kiniry <kiniry@galois.com>
*/

/**
 * The overall shape of the Reactor Trip System (RTS) is an archetypal
 * *sense,compute,actuate* architecture. Sensors are in the `Sensors`
 * subsystem. They are read by the `Instrumentation` subsystem, which
 * contains four separate and independent `Instrumentation`
 * components. The "Compute" part of the architecture is spread across
 * the `Actuation Logic` subsystem,which contains the two `Voting`
 * components which perform the actuation logic itself,and the `Root`
 * subsystem which contains the core computation and I/O components, and
 * the two separate and independent devices that drive actuators.
 */
package id RTS 'Reactor Trip System' {(⊒ RTS (line 6 on page 85), RTS (line 94 on page 87))
  private import 'Semantic Properties'::*;
  import 'Project Glossary'::*;
  import 'RTS Viewpoints and Views'::*;
  import 'RTS Architecture'::*;


  package id Architecture 'RTS Architecture';(⊒ Architecture (line 223 on page 96), Architecture (line 17 on page 85))
  alias Arch for Architecture;
  package id Hardware 'RTS Hardware Artifacts';(⊒ Hardware (line 37 on page 89), RTS Hardware Artifacts (line 1 on page 98), Hardware (line 23 on page 85), Hardware (line 30 on page 91))
  alias HW for Hardware;
  package id Artifacts 'RTS Implementation Artifacts';(⊒ Implementation (line 27 on page 85), Artifacts (line 1 on page 90))
  package id Requirements 'RTS Requirements';(⊒ Requirements (line 31 on page 85))
  package id Properties 'RTS Properties';(⊒ Properties (line 35 on page 86))
  alias Props for Properties;
```

```
  package id Characteristics 'IEEE Std 603,2018 Characteristics';(⊒ Characteristics
       (line 39 on page 86))
  comment TopLevelPackages about Architecture, Hardware, Properties, Characteristics
  /* These are the core top,level subsystems characterizing HARDEN work. */
}
```

D.2 RTS Actions

Listing D.2: Listing SysML Model of RTS_Actions.

```
/**
 * The set of all atomic external or internal actions that the RTS
 * system can take. Note that every scenario must be describable by
 * a sequence of actions.
 */
package id Actions 'RTS Actions' {
  package id Internal 'RTS Internal Actions' {
    action def id IA 'Intenral Action' {
      doc /* Actions internal to the RTS */
    }
    action id Trip 'Signal Trip' : IA {
      in item division;
      in item channel;
    }
    action id Vote 'Vote on Like Trips using Two,out,of,four Coincidence' : IA {(⊒ Vote on
        Like Trips using Two-out-of-four Coincidence (line 57 on page 92))
      in item divisions[2];
      in item channel;
    }
    action id A 'Automatically Actuate Device' : IA {(⊒ Automatically Actuate Device
        (line 60 on page 93))
      in item device;
    }
    action id T 'Self,test of Safety Signal Path' : IA;(⊒ Self-test of Safety Signal Path
        (line 63 on page 93))
  }
  package id External 'RTS External Actions' {
    package 'UI Actions' {
      action def id UIA 'UI Action' {
        doc /* Actions exhibited by the RTS UI, either inbound or outbound. */
      }
      // @design kiniry These should both be specializations of UIA, but I
      // don't know how to write that in SysML yet.
      action def id UI_IA 'UI Input Action';
      action def id UI_OA 'UI Output Action';

      // Input actions.

      action id A Actuate : UI_IA {
        in item actuator;
        in item on_off;
      }
      action id M 'Set Maintenance Mode' : UI_IA {
        in item division;
        in item on_off;
      }
      action id B 'Set Mode' : UI_IA {
        in item division;
        in item trip_mode;
      }
      action id S 'Set Setpoint' : UI_IA {
        in item division;
        in item channel;
```

```
      in item value;
    }
    action id V 'Sensor Value' : UI_IA {
      doc /* Simulate a sensor reading */
      in item division;
      in item channel;
      in item value;
    }
    action id Q 'Quit' : UI_IA;

    // Output actions.
    action id 'Display Pressure' : UI_OA;(⊒ Display Pressure (line 29 on page 92))
    action id 'Display Temperature' : UI_OA;(⊒ Display Temperature (line 32 on
        page 92))
    action id 'Display Saturation Margin' : UI_OA;(⊒ Display Saturation Margin (line 35
        on page 92))
    action id 'Display Trip Ouput Signal State' : UI_OA;
    action id 'Display Indication Of Channel in Bypass' : UI_OA;(⊒ Display Indication of
        Channel in Bypass (line 41 on page 92))
    action id 'Display Actuation State' : UI_OA;
  }
 }
}
```

D.3 RTS Characteristics

Listing D.3: Listing SysML Model of RTS_Characteristics.

```
/**
 * The IEEE 603,2018 requirements (known as "characteristics" in
 * the standard) which the RTS demonstrator system must fulfill.
 */
package id Characteristics 'IEEE Std 603,2018 Characteristics' {(⊒ Characteristics
    (line 39 on page 86))
 requirement def 'Requirements Consistency' {(⊒ Requirements Consistency (line 40 on
     page 102))
   doc /* Requirements must be shown to be consistent. */
 }
 requirement def 'Requirements Colloquial Completeness' {(⊒ Requirements Colloquial
     Completeness (line 47 on page 102))
   doc /* The system must be shown to fulfill all requirements. */
 }
 requirement def 'Requirements Formal Completeness' {(⊒ Requirements Formal Completeness
     (line 52 on page 103))
   doc /* Requirements must be shown to be formally complete. */
 }
 requirement def 'Instrumentation Independence' {(⊒ Instrumentation Independence
     (line 58 on page 103))
   doc /* Independence among the four divisions of instrumentation (inability
          for the behavior of one division to interfere or adversely affect the
          performance of another). */
 }
 requirement def 'Channel Independence' {(⊒ Channel Independence (line 66 on page 103))
   doc /* Independence among the two instrumentation channels within a division
          (inability for the behavior of one channel to interfere or adversely
          affect the performance of another). */
 }
 requirement def 'Actuation Independence' {(⊒ Actuation Independence (line 74 on
     page 103))
   doc /* Independence among the two trains of actuation logic (inability for
          the behavior of one train to interfere or adversely affect the
          performance of another). */
 }
 requirement def 'Actuation Correctness' {(⊒ Actuation Correctness (line 82 on
     page 103))
```

```
    doc /* Completion of actuation whenever coincidence logic is satisfied or
          manual actuation is initiated. */
  }
  requirement def 'Self,Test/Trip Independence' {(⊒ Self-Test/Trip Independence (line 89
      on page 103))
    doc /* Independence between periodic self,test functions and trip functions
          (inability for the behavior of the self,testing to interfere or
          adversely affect the trip functions). */
  }
}
```

D.4 RTS Contexts

Listing D.4: Listing SysML Model of RTS_Contexts.

```
package 'RTS System Contexts' {
  import 'RTS Architecture'::*;
  import 'Kiniry RTS System Architecture Draft'::*;

  // This specification is meant to frame the RTS in the larger context of
  // deployment into a Nuclear Power Plant.

  // Definitions of system contexts.

  // Introduce the RTS context.
  part def 'Reactor Trip System Context';
  alias RTSC for 'Reactor Trip System Context';

  // Introduce the NPP context.
  part def 'Nuclear Power Plant Context';
  alias NPPC for 'Nuclear Power Plant Context';

  part def 'Nuclear Power Plant Operator';
  part def 'Nuclear Power Plant Certifier';

  // Definitions of relevant connections
  connection def 'Digital Instrumentation and Control' {
    end: RTSC[1];
    end: NPPC[1];
  }
  alias DIandC for 'Digital Instrumentation and Control';

  connection def 'NPP Operation' {
    end: 'Nuclear Power Plant Operator'[*];
    end: DIandC[1];
  }

  connection def 'NPP Certification Authority' {
    end: 'NPPC'[1];
    end: 'Nuclear Power Plant Certifier'[1..*];
  }
}
```

D.5 RTS Glossary

Listing D.5: Listing SysML Model of RTS_Glossary.

```
/*
```

```
 # Reactor Trip System (RTS) High,assurance Demonstrator
 ## project: High Assurance Rigorous Digital Engineering for Nuclear Safety (HARDENS)
 ### copyright (C) 2021,2022 Galois
 ### author: Joe Kiniry <kiniry@galois.com>
*/

// @see https://github.com/GaloisInc/HARDENS/issues/30

package id Glossary 'Project Glossary' {(⊒ Glossary (line 1 on page 93))
  // @design Eliminate all redundancy with concepts in KerML or SysML domain
  // libraries.
  private import ScalarValues::*;
  private import KerML::*;

  // Original proposal glossary.
  part def BlueCheck;(⊒ BlueCheck (line 10 on page 93))
  /** A formal, state,based specification language that focuses on the
      specification of the interfaces of discrete modules in a system, and
      often times includes model,based specification constructs to improve
      usability and expressivity. */
  abstract item id BISL 'Behavioral Interface Specification Language';(⊒ BISL (line 4 on
      page 93))
  abstract part def Computer;
  abstract part def Coq;(⊒ Coq (line 15 on page 93))
  abstract part def Cryptol;(⊒ Cryptol (line 25 on page 93))
  abstract item def DevSecOps;(⊒ DevSecOps (line 35 on page 93))
  abstract item def id DIANC 'Digital Instrumentation and Control Systems';(⊒ DIANDC
      (line 43 on page 93))
  /** The NASA Formal Requirements Elicitation Tool is used to make writing,
      understanding, and debugging formal requirements natural and
      intuitive. */
  part def id FRET 'Formal Requirements Elicitation Tool';(⊒ FRET (line 54 on page 94))
  /** An Instruction Set Architecture, or ISA for short, is the set of
      instructions that a given kind of CPU can understand. Example ISAs
      include x86, x64, MIPS, RISC, RISC,V, AVR, etc. */
  attribute def id ISA 'Instruction Set Architecture';
  /** A specification language integrated with support tools and an
      automated theorem prover, developed at the Computer Science Laboratory
      of SRI International. PVS is based on a kernel consisting of an
      extension of Church's theory of types with dependent types, and is
      fundamentally a classical typed higher,order logic. */
  part def PVS;(⊒ PVS (line 59 on page 94))
  /** RISC,V (pronounced ``risk,five'') is an open standard instruction set
      architecture (ISA) based on established reduced instruction set
      computer (RISC) principles. Unlike most other ISA designs, the RISC,V
      ISA is provided under open source licenses that do not require fees to
      use. A number of companies are offering or have announced RISC,V
      hardware, open source operating systems with RISC,V support are
      available and the instruction set is supported in several popular
      software toolchains. */
  attribute def RISC_V_ISA :> ISA;
  /** A formal specification language that uses hierarchical finite state
      machines to specify system requirements. */
  part def id RSML 'Requirements State Modeling Language';(⊒ RSML (line 76 on page 94))
  /** The Boolean satisfiability problem (sometimes called propositional
      satisfiability problem and abbreviated SAT) is the problem of
      determining if there exists an interpretation that satisfies a given
      Boolean formula. */
  abstract item def SAT;(⊒ SAT (line 80 on page 94))
  /** The proof script language is used to specify the assumptions and proof
      goals of formal verifications to the SAW tool. */
  part def SAWscript;(⊒ SAWscript (line 86 on page 94))
  /** A CPU or SoC that is implemented in an HDL and synthesized to a
      bitstream and loaded onto an FPGA. */
  abstract item def 'Soft Core' {
    // size: estimated number of gates
    // complexity: measured complexity metric
```

```
67  |   // hdls: which HDLs are used in the design
68  | }
69  | /** A formally defined computer programming language based on the Ada
70  |     programming language, intended for the development of high integrity
71  |     software used in systems where predictable and highly reliable
72  |     operation is essential. It facilitates the development of applications
73  |     that demand safety, security, or business integrity. */
74  | part def SPARK;(⊒ SPARK (line 94 on page 94))
75  | /** An integrated development environment for formally specifying and
76  |     rigorously analyzing requirements. */
77  | part def SpeAR;(⊒ SpeAR (line 101 on page 94))
78  | /** VCC is a program verification tool that proves correctness of
79  |     annotated concurrent C programs or finds problems in them. VCC extends
80  |     C with design by contract features, like pre, and postcondition as
81  |     well as type invariants. Annotated programs are translated to logical
82  |     formulas using the Boogie tool, which passes them to an automated SMT
83  |     solver Z3 to check their validity. */
84  | part def id VCC 'Verifier for Concurrent C';(⊒ VCC (line 105 on page 94))
85  | /** A software toolchain that includes static analyzers to check
86  |     assertions about a C program; optimizing compilers to translate a C
87  |     program to machine language; and operating systems and libraries to
88  |     supply context for the C program. The Verified Software Toolchain
89  |     project assures with machine,checked proofs that the assertions
90  |     claimed at the top of the toolchain really hold in the
91  |     machine,language program, running in the operating,system context. */
92  | part def id VST 'Verified Software Toolchain';(⊒ VST (line 113 on page 95))
93  |
94  | // Mathematical modeling concepts in RDE.
95  | abstract item def Refinement:> Relationship;(⊒ Refinement (line 122 on page 95))
96  | abstract item def Property:> BooleanExpression;(⊒ Property (line 124 on page 95))
97  | abstract item def 'Safety Property' :> Property;(⊒ Safety Property (line 126 on
        page 95))
98  | abstract item def 'Correctness Property' :> Property;(⊒ Correctness Property (line 129
        on page 95))
99  | abstract item def 'Security Property' :> Property;(⊒ Security Property (line 132 on
        page 95))
100 | abstract item def Model;(⊒ Model (line 135 on page 95))
101 | abstract item def 'Semi,Formal Model' :> Model;(⊒ Semi-Formal Model (line 137 on
        page 95))
102 | abstract item def 'Formal Model' :> Model;(⊒ Formal Model (line 140 on page 95))
103 | abstract item def Consistent :> Property;(⊒ Consistent (line 143 on page 95))
104 | abstract item def Complete :> Property;(⊒ Complete (line 146 on page 95))
105 | abstract item def 'Consistent Model' :> Consistent, Model;(⊒ Consistent Model (line 149
        on page 95))
106 | abstract item def 'Complete Model' :> Complete, Model;(⊒ Complete Model (line 152 on
        page 95))
107 | abstract item def 'Consistent and Complete Model' :> 'Consistent Model', 'Complete Model';
108 | abstract item def Denotational;(⊒ Denotational (line 173 on page 95))
109 | abstract item def Operational;(⊒ Operational (line 175 on page 95))
110 | abstract item def Semantics;(⊒ Semantics (line 177 on page 95))
111 | /** A specification that has a precise, unambiguous, formal semantics
112 |     grounded in real world formal foundations and systems engineering
113 |     artifacts, such as source code and hardware designs. */
114 | abstract item def Rigorous;(⊒ Rigorous (line 187 on page 96))
115 | abstract item def Deterministic;(⊒ Deterministic (line 208 on page 96))
116 | abstract item def 'Non,deterministic';(⊒ Non-deterministic (line 210 on page 96))
117 | abstract part def id FM 'Formal Method';(⊒ FM (line 232 on page 96))
118 |
119 | // Systems modeling concepts in RDE.
120 | // @design Probably in KerML or SysML domain libraries.
121 | abstract item def Requirement;(⊒ Requirement (line 155 on page 95))
122 | abstract item def Scenario;(⊒ Scenario (line 157 on page 95))
123 | abstract item def Product;(⊒ Product (line 159 on page 95))
124 | abstract item def 'Product Line';(⊒ Product Line (line 161 on page 95))
125 | abstract item def Configure;(⊒ Configure (line 163 on page 95))
126 | part def DOORS;(⊒ DOORS (line 165 on page 95))
127 | part def Clafer;(⊒ Clafer (line 167 on page 95))
128 | part def Lobot;(⊒ Lobot (line 169 on page 95))
```

```
129 | abstract item def id FSM 'Finite State Machine';(⊒ FSM (line 206 on page 96))
130 | abstract item def id DFSM 'Deterministic Finite State Machine' (⊒ DFSM (line 212 on
        page 96))
131 |   :> FSM, Deterministic;
132 | abstract item def id NFSM 'Non,deterministic Finite State Machine' (⊒ NFSM (line 215 on
        page 96))
133 |   :> FSM, 'Non,deterministic';
134 | abstract item def id ASM 'Abstract State Machine';(⊒ ASM (line 218 on page 96))
135 | abstract part def Design;(⊒ Design (line 221 on page 96))
136 | abstract part def Architecture;(⊒ Architecture (line 223 on page 96), Architecture
        (line 17 on page 85))
137 | abstract part def Specification;(⊒ Specification (line 225 on page 96))
138 | abstract part def 'Architecture Specification' :> Specification;(⊒ Architecture
        Specification (line 227 on page 96))
139 | abstract part def System;(⊒ System (line 255 on page 97))
140 | abstract part def 'Distributed System' :> System;(⊒ Distributed System (line 257 on
        page 97))
141 | abstract part def 'Concurrent System' :> System;(⊒ Concurrent System (line 260 on
        page 97))
142 | abstract part def Algorithm;
143 | abstract part def Program;(⊒ Program (line 290 on page 97))
144 |
145 | // Concepts related to measurable abstractions of systems.
146 | abstract item def Risk;(⊒ Risk (line 179 on page 96))
147 | abstract item def Power;(⊒ Power (line 181 on page 96))
148 | abstract item def Resource;(⊒ Resource (line 183 on page 96))
149 | abstract item def Reliability;(⊒ Reliability (line 185 on page 96))
150 |
151 | // Assurance concepts and techonlogies.
152 | abstract item def id CDE 'Collaborative Development Environment';(⊒ CDE (line 192 on
        page 96))
153 | abstract item def id CI 'Continuous Integration';(⊒ CI (line 194 on page 96))
154 | abstract item def id CV 'Continuous Verification';(⊒ CV (line 22 on page 86), CV
        (line 196 on page 96))
155 | abstract item def Analyzer;(⊒ Analyzer (line 198 on page 96))
156 | abstract item def 'Static Analyzer' :> Analyzer;(⊒ Static Analyzer (line 200 on
        page 96))
157 | abstract item def 'Dynamic Analyzer' :> Analyzer;(⊒ Dynamic Analyzer (line 203 on
        page 96))
158 | abstract part def Solver;(⊒ Solver (line 230 on page 96))
159 | abstract part def id LF 'Logical Framework';(⊒ LF (line 234 on page 96))
160 | abstract item def 'High,Assurance';(⊒ High-Assurance (line 308 on page 98))
161 |
162 | // Concepts relevant to languages and protocols.
163 | abstract part def Language;
164 | abstract part def 'Specification Language' :> Language;(⊒ Specification Language
        (line 238 on page 96))
165 | abstract part def Protocol;(⊒ Protocol (line 240 on page 96))
166 | abstract part def 'System Specification' :> Specification;(⊒ System Specification
        (line 242 on page 96))
167 | abstract item def 'Hand,written';(⊒ Hand-written (line 245 on page 97))
168 | abstract item def 'Machine,generated';(⊒ Machine-generated (line 247 on page 97))
169 | abstract part def 'Source,level Specification Language' (⊒ Source-level Specification
        Language (line 249 on page 97))
170 |   :> 'Specification Language';
171 | abstract part def 'Model,based Specification Language' (⊒ Model-based Specification
        Language (line 252 on page 97))
172 |   :> 'Specification Language';
173 | abstract item def Cryptological;
174 | abstract item def 'Cryptographic Protocol' :> Protocol, Cryptological;(⊒ Cryptographic
        Protocol (line 263 on page 97))
175 | abstract item def 'Cryptographic Algorithm' :> Algorithm, Cryptological;(⊒ Cryptographic
        Algorithm (line 266 on page 97))
176 |
177 | // Software engineering.
178 | abstract item def id PL 'Programming Language' :> Language;(⊒ PL (line 236 on page 96))
179 | abstract item def 'Source Code';
180 | abstract part def C :> 'PL';(⊒ C (line 310 on page 98))
181 | abstract part def C_Source :> C, 'Source Code';
182 | abstract item def 'Object Code';
```

```
abstract item def id IR 'Intermediate Representation';(⊒ IR (line 66 on page 87))
abstract item def id LLVM 'Low,Level Virtual Machine' :> IR;(⊒ LLVM (line 70 on page 87))
abstract item def Compiler {(⊒ Compiler (line 280 on page 97))
  item input: Language[1..*];
  item output: Language[1..*];
}

// Hardware design.
abstract item def Hardware;(⊒ Hardware (line 37 on page 89), Hardware (line 23 on page 85), Hardware (line 30 on page 91))
abstract item def SWaP {(⊒ SWaP (line 112 on page 88))
  // attribute size:
  // attribute weight:
  // attribute power:
}
abstract item def Hard :> SWaP;
abstract item def 'Soft Core Hardware' :> Hardware, 'Soft Core';
abstract item def 'Physical Hardware' :> Hardware, Hard;
abstract part def Synthesizer :> Compiler;(⊒ Synthesizer (line 282 on page 97))
abstract item def id HDL 'Hardware Design Language';(⊒ HDL (line 50 on page 87))
abstract part def BluespecSystemVerilog :> HDL;
abstract part def SystemVerilog :> HDL;(⊒ SV (line 108 on page 88))
abstract part def Verilog :> SystemVerilog;(⊒ Verilog (line 134 on page 88))
abstract part def Chisel :> HDL;
abstract part def id CPU 'Central Processing Unit';(⊒ CPU (line 38 on page 91), CPU (line 19 on page 86))

// Hardware engineering concepts.
abstract part def Component;
abstract part def Switch :> Component;
abstract part def Button :> Component;
abstract part def Header :> Component;
abstract part def Interface :> Component;
abstract part def Connector :> Component;(⊒ Connector (line 301 on page 97))
abstract part def Memory :> Component;
abstract part def ASIC :> Component;(⊒ ASIC (line 13 on page 86))
abstract item def id IO 'I/O';(⊒ IO (line 268 on page 97))
abstract part def id GPIO 'General Purpose I/O';(⊒ GPIO (line 46 on page 87), GPIO (line 270 on page 97))
abstract part def Sensor;(⊒ Sensors (line 49 on page 89), Sensor (line 273 on page 97))
abstract part def 'Temperature Sensor';(⊒ Temperature Sensor 1 (line 51 on page 89), Temperature Sensor (line 46 on page 99), Temperature Sensor 2 (line 53 on page 89), TS1 (line 84 on page 99), TS2 (line 87 on page 99))
abstract part def 'Pressure Sensor';(⊒ Pressure Sensor 2 (line 57 on page 89), Pressure Sensor (line 50 on page 99), PS2 (line 93 on page 100), PS1 (line 90 on page 100), Pressure Sensor 1 (line 55 on page 89))
abstract part def Actuator;(⊒ Actuator 2 (line 47 on page 89), Actuator 1 (line 45 on page 89), Actuator (line 275 on page 97), Actuators (line 43 on page 89))
abstract part def Solenoid :> Actuator;(⊒ Solenoid (line 277 on page 97))
abstract item def Bus;
abstract part def id USB 'Universal Serial Bus' :> Bus;(⊒ USB (line 124 on page 88), USB (line 284 on page 97))
abstract part def LED;(⊒ LED (line 286 on page 97))
abstract part def Cable;(⊒ Cable (line 288 on page 97))
abstract part def id FPGA 'Field,Programmable Gate Array' :> ASIC;(⊒ FPGA (line 38 on page 86), FPGA (line 294 on page 97), FPGA (line 39 on page 89))
abstract part def 'ECP,5' :> FPGA;(⊒ ECP-5 (line 296 on page 97))
abstract part def id PCB 'Printed Circuit Board' {(⊒ PCB (line 299 on page 97))
  part components: Component[*];
}
abstract part def 'USB Connector' :> USB, Connector;(⊒ USB Connector (line 303 on page 97))
abstract part def id USB_Mini 'USB Mini Connector' :> 'USB Connector';(⊒ USB-Mini (line 305 on page 98))
abstract part def PMOD;(⊒ PMOD (line 312 on page 98))
abstract part def JTAG :> Protocol;(⊒ JTAG (line 314 on page 98))
abstract part def Driver;(⊒ Driver (line 316 on page 98))
```

```
  port def USB_In {
    in item 'USB Connector';
  }
  port def USB_Out {
    out item 'USB Connector';
  }
  /** A normal USB cable. */
  abstract part def 'USB Cable' :> USB, Cable {(⊒ USB Cable (line 5 on page 98), UCB
      Cable (line 320 on page 98))
    /** What kind of USB connector is on the start of the cable? */
    port start_connector: USB_In;
    /** What kind of USB connector is on the end of the cable? */
    port end_connector: USB_Out;
  }
  port def 'Output LED' :> LED;

  // Safety,critical concepts.
  abstract item def Voting;(⊒ Voting (line 318 on page 98), Voting 2 (line 23 on
      page 89), Voting 1 (line 21 on page 89))

  // Artifacts specific to RDE.
  abstract part def id CryptolSpec 'Cryptol System Specification' (⊒ CryptolSpec (line 3
      on page 90))
   :> Cryptol, 'System Specification' {
     attribute literate: Boolean;
   }
  attribute def Languages {
    attribute languages: String[*];
  }
  abstract part def id Impl 'Implementation' {
    attribute languages: Languages[*];
  }
  abstract part def id Software 'Software Implementation'(⊒ Software (line 22 on
      page 90))
    :> Implementation;
  abstract part def id SWImpl 'Hand,written Software Implementation' (⊒ HWImpl (line 32
      on page 91), SWImpl (line 24 on page 90))
    :> Software, 'Hand,written';
  abstract part def id SynthSW 'Synthesized Software Implementation' (⊒ SynthSW (line 27
      on page 90), SynthHW (line 35 on page 91))
    :> Software, 'Machine,generated';
  abstract part def 'Hardware Implementation';(⊒ Hardware (line 30 on page 91))
  abstract part def id HWImpl 'Hand,written Hardware Implementation';(⊒ HWImpl (line 32
      on page 91), SWImpl (line 24 on page 90))
  abstract part def id SynthHW 'Synthesized Hardware Implementation';(⊒ SynthSW (line 27
      on page 90), SynthHW (line 35 on page 91))
  abstract part def id Binary 'Software Binaries' {(⊒ Binaries (line 46 on page 91))
    attribute verified_compilation: Boolean;
    attribute secure_compilation: Boolean;
    attribute isa: ISA;
  }
  part def RISCV_Binary :> Binary {
    // :>> isa = RISC_V_ISA;
  }
  abstract part def id Bitstream 'FPGA Bitstream' {(⊒ Bitstream (line 50 on page 91),
      Bitstream (line 292 on page 97))
    attribute proprietary_flow: Boolean;
  }

  // NRC concepts.
  abstract part def 'NRC Certifcation Regulations';
}
```

D.6 RTS Hardware Artifacts

Listing D.6: Listing SysML Model of RTS_Hardware_Artifacts.

```
/**
 * The physical hardware components that are a part of the HARDENS RTS
 * demonstrator.
 */
package 'RTS Hardware Artifacts' {(⊒ RTS Hardware Artifacts (line 1 on page 98),
      Hardware (line 23 on page 85))
  private import 'Project Glossary'::*;
  //import Architecture::RTS_System_Arch::Hardware::*;
  private import ScalarValues::*;

  part def 'SERDES Test SMA Connector' :> Connector;(⊒ J9-J26 (line 12 on page 98))
  part def 'Parallel Config Header' :> Header;(⊒ J38 (line 14 on page 98))
  part def 'Versa Expansion Connector' :> Connector;(⊒ J39-J40 (line 16 on page 98))
  part def 'SPI Flag Configuration Memory' :> Memory;(⊒ U4 (line 18 on page 98))
  part def 'CFG Switch' :> Switch;
  part def 'Input Switch' :> Switch;(⊒ SW5 (line 22 on page 98))
  part def 'Output LED' :> LED;(⊒ D5-D12 (line 24 on page 98), Output LED (line 323
       on page 98))
  part def 'Input Push Button' :> Button;(⊒ SW2-SW4 (line 26 on page 98))
  part def '12 V DC Power Input' :> Power;
  part def 'GPIO Headers' :> Header, GPIO;(⊒ J32-J33 (line 30 on page 98))
  part def 'PMOD/GPIO Header' :> Header, PMOD, GPIO;(⊒ J31 (line 32 on page 99))
  part def 'Microphone Board/GPIO Header' :> Header;(⊒ J30 (line 34 on page 99))
  part def 'ECP5,5G Device' :> FPGA;(⊒ U3 (line 38 on page 99))

  part def 'JTAG Interface' :> JTAG, USB;(⊒ J1 (line 40 on page 99))
  part def 'Mini USB Programming' :> USB;(⊒ J2 (line 42 on page 99))
  part def id DevBoard 'Lattice ECP,5 FPGA Development Board' :> PCB {(⊒ Lattice ECP-5 FGPA
       Development Board (line 41 on page 89), Board (line 44 on page 99))
    part J9_J26 : 'SERDES Test SMA Connector'[16] subsets components;
    part J38 : 'Parallel Config Header' subsets components;(⊒ J38 (line 14 on page 98))
    part J39_J40 : 'Versa Expansion Connector'[2] subsets components;
    part U4 : 'SPI Flag Configuration Memory' subsets components;(⊒ U4 (line 18 on
         page 98))
    part SW1 : 'CFG Switch' subsets components;(⊒ SW1 (line 20 on page 98))
    part SW5 : 'Input Switch' subsets components;(⊒ SW5 (line 22 on page 98))
    part D5_D12 : 'Output LED'[8] subsets components;
    part SW2_SW4 : 'Input Push Button'[3] subsets components;
    part J37 : '12 V DC Power Input' subsets components;
    part J5_J8_J32_J33 : 'GPIO Headers'[4] subsets components;
    part J31 : 'PMOD/GPIO Header' subsets components;(⊒ J31 (line 32 on page 99))
    part J30 : 'Microphone Board/GPIO Header' subsets components;(⊒ J30 (line 34 on
         page 99))
    part 'Prototype Area';(⊒ Prototype Area (line 36 on page 99))
    part U3 : 'ECP5,5G Device' subsets components;(⊒ U3 (line 38 on page 99))
    part J1 : 'JTAG Interface' subsets components;(⊒ J1 (line 40 on page 99))
    part J2 : 'Mini USB Programming' subsets components;(⊒ J2 (line 42 on page 99))
  }

  enum def SolenoidState {
    OPEN;
    CLOSED;
  }
  /** A solenoid actuator capable of being in an open or closed state. */
  part def 'Solenoid Actuator' :> Actuator {(⊒ SA1 (line 96 on page 100), SA2
       (line 99 on page 100), Solenoid Actuator (line 54 on page 99))
    item actuator_state;
    /** Open! */
    port open;
    /** Close! */
    port close;
  }
}
```

D.7 RTS Implementation Artifacts

Listing D.7: Listing SysML Model of RTS_Implementation_Artifacts.

```
package id Artifacts 'RTS Implementation Artifacts' {(⊒ Implementation (line 27 on page 85), Artifacts (line 1 on page 90))
  private import ScalarValues::*;
  private import 'Project Glossary'::*;

  // @design Remove concepts in general Glossary that duplicate or
  // overlap with these concepts. Move abstract items to Glossary.
  part def id CryptolToC 'Cryptol Software Compiler' :> Compiler {(⊒ CryptolToC (line 12 on page 90))
    ref item input: CryptolSpec redefines input;
    ref item output: C_Source redefines output;
  }
  part def id CryptolToSystemVerilog 'Cryptol Hardware Compiler' :> Compiler {(⊒ CryptolToSystemVerilog (line 17 on page 90))
    ref item input: CryptolSpec redefines input;
    ref item output: SystemVerilog redefines output;
  }
  part def id CPU 'COTS High,Assurance RV32I RISC,V CPU' :> CPU, RISC_V_ISA;(⊒ CPU (line 38 on page 91), CPU (line 19 on page 86))
  part def id CompCert 'CompCert Compiler' :> Compiler {(⊒ CompCert (line 40 on page 91))
    ref item input: C_Source redefines input;
    ref item output: RISCV_Binary redefines output;
  }
  part def id BSC 'Bluespec Compiler' :> Compiler {(⊒ BSC (line 42 on page 91))
    ref item input: BluespecSystemVerilog redefines input;
    ref item output: SystemVerilog redefines output;
  }
  part def id SymbiFlow 'SymbiFlow Synthesizer' :> Synthesizer {(⊒ SymbiFlow (line 44 on page 91))
    ref item input: SystemVerilog redefines input;
    ref item output: Bitstream redefines output;
  }
  part def id RTL 'Demonstrator Verilog';(⊒ RTL (line 92 on page 87), RTL (line 48 on page 91))
  part def 'Demonstrator Bitstream' :> Bitstream;
  package id Dataflow 'Dataflow of RTS Implementation Artifacts' {(⊒ Dataflow (line 52 on page 91))
    private import 'RTS Implementation Artifacts'::*;

    part def 'HARDENS Cryptol System Specification' :> CryptolSpec {
     // :>> literate = true;
    }
    // bind 'HARDENS Cryptol System Specification'.output = CryptolToC.input;
  }
}
```

D.8 RTS Physical Architecture

Listing D.8: Listing SysML Model of RTS_Physical_Architecture.

```
///** The physical architecture of the HARDENS RTS demonstrator. */
package 'Physical Architecture' {(⊒ Physical Architecture (line 63 on page 99))
  import 'Project Glossary'::*;
  import 'RTS Hardware Artifacts'::*;

// /** A PCB developer board used to prototype hardware. */
// part 'HARDENS Demonstrator Board' : DevBoard;
// /** The USB cable used to communicate the ASCII UI to/from the board. */
```

```
// part id UI_C 'USB UI Cable' : 'USB Cable';
// /** The USB cable used to program the board with a bitstream. */
// part id Prog_C 'USB Programming Cable' : 'USB Cable';
// /** The USB cable used to interact with the board in a debugger. */
// part id Debug_C 'USB Debugging I/O Cable' : 'USB Cable';
// // @trace #11 https://github.com/GaloisInc/HARDENS/issues/11

// part def id MPL3115A2 'SparkFun Altitude/Pressure Sensor Breakout' :>
// PCB, 'Pressure Sensor';
// // 4x https://www.sparkfun.com/products/11084
// part def 'SparkFun MOSFET Power Control Kit' :> PCB, Power;
// // 4x https://www.sparkfun.com/products/12959
// part def id TMP102 'SparkFun Digital Temperature Sensor Breakout' :>
// PCB, 'Temperature Sensor';
// // 4x https://www.sparkfun.com/products/13314
// part def 'Small Push,Pull Solenoid ,12VDC' :> 'Solenoid Actuator';
// // 4x https://www.adafruit.com/product/412
// part def '1N4001 Diode';
// // 1x https://www.adafruit.com/product/755
// /** The first of two redundant temperature sensors. */
// part id TS1 'Temperature Sensor 1' : TMP102;
// /** The second of two redundant temperature sensors. */
// part id TS2 'Temperature Sensor 2' : TMP102;
// /** The first of two redundant pressure sensors. */
// part id PS1 'Pressure Sensor 1' : MPL3115A2;
// /** The second of two redundant pressure sensors. */
// part id PS2 'Pressure Sensor 2' : MPL3115A2;
// /** The first of two redundant solenoid actuators. */
// part id SA1 'Solenoid Actuator 1' : 'Small Push,Pull Solenoid ,12VDC';
// /** The second of two redundant solenoid actuators. */
// part id SA2 'Solenoid Actuator 2' : 'Small Push,Pull Solenoid ,12VDC';

//
// /** The computer used by a developer to interface with the demonstrator,
// typically for driving the demonstrator's UI and programming and
// debugging the board. */
// part def 'Developer Machine':> Computer;
//
// /** The fully assembled HARDENS demonstrator hardware with all component present. */
// part id Demonstrator 'HARDENS Demonstrator';
//
// connection def DevMachineToDevBoard {
// end: Computer;
// end: PCB;
// }
// connection: DevMachineToDevBoard connect 'Developer Machine' to Board;
}
```

D.9 RTS Properties

Listing D.9: Listing SysML Model of RTS_Properties.

```
/**
 * All correctness and security properties of the RTS system are
 * specified in this subsystem.
 */
package id Properties 'RTS Properties' {(□  Properties   (line 35 on page 86))
}
```

D.10 RTS Requirements

Listing D.10: Listing SysML Model of RTS_Requirements.

```
/**
 * All requirements that the RTS system must fulfill, as driven by the
 * IEEE 603,2018 standards and the NRC RFP.
 */
package id Requirements 'RTS Requirements' {(⊒ Requirements (line 31 on page 85))
  // Note that we do not specify documentation comments here as they
  // are specified in the Lando specification. If we do not include
  // additional specifications here on the refinement from the higher,level
  // specification (in this case, SysML refines Lando), then the higher,level
  // specification's comments/specifications refine too (an hence are
  // just copied verbatim).
  package id Requirements 'HARDENS Project High,level Requirements' {(⊒ Requirements
      (line 31 on page 85))
    import 'Project Glossary'::*;
    import 'RTS Stakeholders'::*;

    requirement def 'Project Requirements' {
      subject 'NRC staff' : 'NRC Customer';
    }
    requirement 'NRC Understanding' : 'Project Requirements';(⊒ NRC Understanding (line 7
        on page 102))
    requirement 'Identify Regulatory Gaps' : 'Project Requirements';(⊒ Identify Regulatory
        Gaps (line 13 on page 102))
    requirement Demonstrate : 'Project Requirements';(⊒ Demonstrate (line 18 on
        page 102))
    requirement 'Demonstrator Parts' : 'Project Requirements';(⊒ Demonstrator Parts
        (line 24 on page 102))
    requirement 'Demonstrator Groundwork' : 'Project Requirements';(⊒ Demonstrator
        Groundwork (line 28 on page 102))
}

// those found in the Characteristics specification.
package id Characteristics 'NRC Characteristics' {(⊒ Characteristics (line 39 on
    page 86))
  import 'Project Glossary'::*;
  import 'RTS Stakeholders'::*;

  requirement def 'NRC Characteristic' {
    //subject expert: 'NRC Certification SME';
    //subject regulation: 'NRC Certification Regulations';
  }
  requirement 'Requirements Consistency' : 'NRC Characteristic';(⊒ Requirements
      Consistency (line 40 on page 102))
  requirement 'Requirements Colloquial Completeness' : 'NRC Characteristic';(⊒
      Requirements Colloquial Completeness (line 47 on page 102))
  requirement 'Requirements Formal Completeness' : 'NRC Characteristic';(⊒ Requirements
      Formal Completeness (line 52 on page 103))
  requirement 'Instrumentation Independence' : 'NRC Characteristic';(⊒ Instrumentation
      Independence (line 58 on page 103))
  requirement 'Channel Independence' : 'NRC Characteristic';(⊒ Channel Independence
      (line 66 on page 103))
  requirement 'Actuation Independence' : 'NRC Characteristic';(⊒ Actuation Independence
      (line 74 on page 103))
  requirement 'Actuation Correctness' : 'NRC Characteristic';(⊒ Actuation Correctness
      (line 82 on page 103))
  requirement 'Self,Test/Trip Independence' : 'NRC Characteristic';(⊒ Self-Test/Trip
      Independence (line 89 on page 103))
}
// Note that formal requirements expressed externally must be traceable
// to this system model, but need not be repeated in whole here. Model
// elements that are expressed in both the SysML and FRET models must
// be in a refinement relationship with each other (e.g,, in this case study,
// SysML ⊑ FRET.
package 'Formal Requirements' {
  import 'Project Glossary'::*;
```

```
51    requirement def id FRET 'FRET Requirements' {(⊒  FRET   (line 54 on page 94))
52      doc /* RTS requirements formalized in the FRET tool. */
53    }
54
55    requirement ACTUATION_ACTUATOR_0 : FRET;
56  }
57 }
```

D.11 RTS Scenarios

Listing D.11: Listing SysML Model of RTS_Scenarios.

```
1  /**
2   * The set of all scenarios that describe interesting end,to,end executions
3   * of the RTS system. The full set of scenarios must include all normal
4   * behavior (online and during self,test) and exceptional behavior.
5   */
6  package id Scenarios 'RTS Scenarios' {
7    package id Normal 'RTS Normal Behavior Scenarios' {
8      import 'RTS Architecture'::'RTS System Architecture'::RTS;
9
10     item def 'RTS User';
11     use case def id NB 'Normal Behavior' {
12       subject RTS;
13       actor user : 'RTS User';
14       objective {
15         doc /* @see test_scenarios.lando */
16       }
17     }
18     // @design kiniry Actually, ST should specialize NB. I don't know how
19     // to express that in SysML yet.
20     use case def id ST 'Normal Behavior Under Self,Test' {
21         subject RTS;
22         actor tester : 'RTS User';
23     }
24     package 'Self,test Scenarios' {
25       import Normal::ST;
26       use case '1a ,Trip on Mock High Pressure Reading from that Pressure Sensor' : NB;
27       use case '1b ,Trip on Environmental High Pressure Reading from that Pressure Sensor' :
             ↪ NB;
28       use case '2a ,Trip on Mock High Temperature Reading from that Temperature Sensor' : NB;
29       use case '2a ,Trip on Environmental High Temperature Reading from that Temperature
             ↪ Sensor' : NB;
30       use case '3a ,Trip on Mock Low Saturation Margin' : NB;
31       use case '3a ,Trip on Environmental Low Saturation Margin' : NB;
32       use case '4 ,Vote on Every Possible Like Trip' : NB;
33       use case '5a ,Automatically Actuate All Mock Devices in Sequence' : NB;
34       use case '5b ,Automatically Actuate All Mock Devices in Sequence' : NB;
35       use case '6 ,Manually Actuate Each Device in Sequence' : NB;
36       use case '7a ,Select Maintenance Operating Mode for each Division' : NB;
37       use case '7b ,Select Normal Operating Mode for each Division' : NB;
38       use case '8 ,Perform Each Kind of Setpoint Adjustment' : NB;
39       use case '9 ,Configure Bypass of Each Instrument Channel in Sequence' : NB;
40       use case '10 ,Configure Active Trip Output State of Each Instrument Channel in Sequence
             ↪ ' : NB;
41       use case '11 ,Display Pressure, Temperature, and Saturation Margin' : NB;
42       use case '13 ,Display Indication of Every Channel in Bypass in Sequence' : NB;
43       use case '14 ,Demonstrate Periodic Continual Self,test of Safety Signal Path' : NB;
44       use case 'Full Self,Test' : NB;
45     }
46     package 'RTS Scenarios' {
47
```

```
    }
  }
  package id Exceptional 'RTS Exceptional Behavior Scenarios' {
    use case def id EB 'Exceptional Behavior' {
      subject RTS;
      objective {
        doc /* @see test_scenarios.lando */
      }
    }
    use case '1a ,Cause Actuator 1 to Fail' : EB;
    use case '1b ,Cause Actuator 2 to Fail' : EB;
    use case '1c ,Non,determinisitically Cause an Actuator to Eventually Fail' : EB;
    use case '2a ,Cause Temperature Sensor 1 to Fail' : EB;
    use case '2b ,Cause Temperature Sensor 2 to Fail' : EB;
    use case '2c ,Non,deterministically Cause a Temperature Sensor to Eventually Fail' : EB;
    use case '3a ,Cause Pressure Sensor 1 to Fail' : EB;
    use case '3b ,Cause Pressure Sensor 2 to Fail' : EB;
    use case '3c ,Non,deterministically Cause a Pressure Sensor to Eventually Fail' : EB;
    use case '4a ,Cause Instrumentation Unit 1 to Fail' : EB;
    use case '4b ,Cause Instrumentation Unit 2 to Fail' : EB;
    use case '4c ,Cause Instrumentation Unit 3 to Fail' : EB;
    use case '4d ,Cause Instrumentation Unit 4 to Fail' : EB;
    use case '4e ,Non,Deterministically Cause Instrumentation Unit to Eventually Fail' : EB;
    // @review kiniry I actually don't know if we are fault tolerant to' : EB;
    // failure in these components. Please review @abakst.
    use case '5a ,Cause Temperature Demultiplexor 1 to Fail' : EB;
    use case '5b ,Cause Temperature Demultiplexor 2 to Fail' : EB;
    use case '5b ,Cause a Temperature Demultiplexor to Eventualy Fail' : EB;
  }
}
```

D.12 RTS Stakeholders

Listing D.12: Listing SysML Model of RTS_Stakeholders.

```
package 'RTS Stakeholders'
{
  // NRC Stakeholders
  part def 'NRC Customer';
  part def 'NRC Assurance SME' :> 'NRC Customer';
  part def 'NRC Certification SME' :> 'NRC Customer';

  // Galois Stakeholders
  part def 'Galois Employee' {
    attribute
      name: ScalarValues::String;
      email: ScalarValues::String;
  }
  part def 'Galois PI' :> 'Galois Employee';
  part def 'Galois PL' :> 'Galois Employee';
  part def 'Galois Research Engineer' :> 'Galois Employee';
  part def 'Galois Software and Assurance Research Engineer'
    :> 'Galois Research Engineer';
  part def 'Galois Hardware Research Engineer' :> 'Galois Research Engineer';
  part PI : 'Galois PI' {
    attribute
      redefines name = "Joe Kiniry";
      redefines email = "kiniry@galois.com";
  }

  part PL : 'Galois PL' {(⊒  PL   (line 236 on page 96))
    attribute
```

```
28        redefines name = "Andrew Bivin";
29        redefines email = "abivin@galois.com";
30    }
31    part SARE : 'Galois Software and Assurance Research Engineer' {
32      attribute
33        redefines name = "Alex Bakst";
34        redefines email = "abakst@galois.com";
35    }
36    part HRE : 'Galois Hardware Research Engineer' {
37      attribute
38        redefines name = "Michal Podhradsky";
39        redefines email = "mpodhradsky@galois.com";
40    }
41
42    concern 'requirement traceability' {
43      doc /* Will all requirements be traceable from project requirement
44           * or NRC characteristic to assurance evidence demonstrated in a
45           * report? */
46      stakeholder 'Customer';
47      stakeholder 'Galois Employee';
48    }
49
50
51    // or from Galois or the Galois HARDENS team.
52  }
```

D.13 RTS Static Architecture

Listing D.13: Listing SysML Model of RTS_Static_Architecture.

```
1   /**
2    * This RTS architecture specification includes all of the core
3    * concepts inherent to NPP Instrumentation and Control systems.
4    * A system architecture specification often includes a software,
5    * hardware, network, and data architecture specifications.
6    */
7   package id Architecture 'RTS Architecture' {(⊒ Architecture (line 223 on page 96),
        Architecture (line 17 on page 85))
8     //import RTS::*;
9     //import 'Project Glossary'::*;
10    //import Artifacts::*;
11    //import 'RTS Hardware Artifacts'::*;
12
13    part def Base_RTS_System_Architecture_Context {
14      // overall RTS architecture shape from
15    }
16
17    /** Note that this is the *systems* architecture, which is different
18        than our software, hardware, or data architectures. */
19    package id RTS_System_Arch 'RTS System Architecture' {(⊒ RTS_System_Arch (line 3 on
          page 88))
20
21      package Sensor {(⊒ Sensors (line 49 on page 89), Sensor (line 273 on page 97))
22        private import Quantities::*;
23
24        /** Generic sensor port */
25        port def SensorOutPort {
26            out value : ScalarQuantityValue;
27        }
28
29        /** Generic sensor */
30        part def GenericSensor {
31          attribute currentValue : ScalarQuantityValue;
```

```
32        attribute sensorAddress : ScalarValues::Integer;
33        port output: SensorOutPort;
34      }
35
36      /**
37       * A demultiplexer for sending one sensor signal to multiple
38       * outputs.
39       */
40      part def Demux {
41        port input: ~SensorOutPort;
42        // Using vector notation doesn't seem to work in connections
43        port output1: SensorOutPort;
44        port output2: SensorOutPort;
45      }
46    }
47
48    /** A generic temperature sensor. */
49    package TempSensor {
50      import Sensor::*;
51      import ISQThermodynamics::TemperatureValue;
52
53      /** Temperature port */
54      port def TemperatureOutPort :> SensorOutPort {
55          redefines value: TemperatureValue;
56      }
57
58      /** A sensor that is capable of measuring the temperature of its environment. */
59      part def 'Temperature Sensor' :> GenericSensor {(⊒  Temperature Sensor 1   (line 51 on
            page 89),   Temperature Sensor   (line 46 on page 99),   Temperature Sensor 2
            (line 53 on page 89),   TS1   (line 84 on page 99),   TS2   (line 87 on page 99))
60        /** What is your temperature reading in Celsius (C)? */
61        redefines currentValue: TemperatureValue;
62        redefines output: TemperatureOutPort;
63      }
64
65      part def TempDemux :> Demux {
66        redefines input: ~TemperatureOutPort;
67        redefines output1: TemperatureOutPort;
68        redefines output2: TemperatureOutPort;
69      }
70    }
71
72    /** A generic pressure sensor. */
73    package PressureSensor {(⊒  Pressure Sensor 2   (line 57 on page 89),   Pressure Sensor
          (line 50 on page 99),   PS2   (line 93 on page 100),   PS1   (line 90 on page 100),
          Pressure Sensor 1   (line 55 on page 89))
74      import Sensor::*;
75      import ISQMechanics::PressureValue;
76
77      /** Pressure port */
78      port def PressureOutPort :> SensorOutPort {
79          redefines value: PressureValue;
80      }
81
82      /** A sensor that is capable of measuring the air pressure of its environment. */
83      part def 'Pressure Sensor' :> GenericSensor {(⊒  Pressure Sensor 2   (line 57 on
            page 89),   Pressure Sensor   (line 50 on page 99),   PS2   (line 93 on
            page 100),   PS1   (line 90 on page 100),   Pressure Sensor 1   (line 55 on
            page 89))
84        /** What is your pressure reading in Pascal (P)? */
85        redefines currentValue: PressureValue;
86        redefines output: PressureOutPort;
87      }
88
89      part def PressureDemux :> Demux {
90        redefines input: ~PressureOutPort;
91        redefines output1: PressureOutPort;
92        redefines output2: PressureOutPort;
```

```
93        }
94      }
95
96      /**
97       * The Instrumentation subsystem contains all of the sensors for an
98       * NPP I&C system.
99       */
100     package Instrumentation {(⊒ Instrumentation 3 (line 65 on page 89), Instrumentation
            2 (line 63 on page 89), Instrumentation 1 (line 61 on page 89),
            Instrumentation (line 59 on page 89), Instrumentation 4 (line 67 on
            page 89))(⊑ Instrumentation (?? on page ??))
101       private import ScalarValues::Real;
102       private import ScalarValues::Boolean;
103       private import TempSensor::*;
104       private import PressureSensor::*;
105
106       port def TripPort {
107         out trip : Boolean;
108       }
109
110       port def BypassPort {
111         out trip : Boolean;
112       }
113
114       enum def TripMode {
115         enum Bypass;
116         enum Operate;
117         enum Manual;
118       }
119
120       enum def Channel {
121         enum Temperature;
122         enum Pressure;
123         enum Saturation;
124       }
125
126       attribute def TripModeCommand {
127         attribute mode: TripMode;
128         attribute channel: Channel;
129       }
130
131       port def TripModePort {
132         out mode: TripModeCommand;
133       }
134
135       part def InstrumentationUnit {
136         // setpoints
137         attribute tempSetpoint : TemperatureValue;
138         attribute pressureSetpoint : PressureValue;
139         attribute saturationLimit : Real;
140
141         // mode selectors
142         attribute maintenanceMode : Boolean;
143         attribute temperatureTripMode: TripMode;
144         attribute pressureTripMode: TripMode;
145         attribute saturationTripMode: TripMode;
146
147
148         // Inputs
149         port temperatureInput: ~TemperatureOutPort;
150         port pressureInput: ~PressureOutPort;
151         port tripMode: ~TripModePort;
152
153         // Outputs
154         port pressureTripOut:TripPort;
155         port temperatureTripOut:TripPort;
156         port saturationTripOut:TripPort;
```

```
157
158         port setMaintenanceMode: ~EventControl::MaintenancePort;
159
160         port newTemperatureSetpoint: ~TemperatureOutPort;
161         port newPressureSetpoint: ~PressureOutPort;
162         port newSaturationSetpoint : ~SensorOutPort;
163       }
164     }
165
166     package Actuation {(⊑ Actuation (?? on page ??))
167       import Instrumentation::*;
168
169       port def ActuationPort {
170         out actuate: ScalarValues::Boolean;
171       }
172
173       part def CoincidenceLogic {
174         port channel1: ~TripPort;
175         port channel2: ~TripPort;
176         port channel3: ~TripPort;
177         port channel4: ~TripPort;
178         port actuate: ActuationPort;
179       }
180       part def OrLogic {
181         port channel1: ~TripPort;
182         port channel2: ~TripPort;
183         port actuate: ActuationPort;
184       }
185
186      part def ActuationUnit {
187        part temperatureLogic : CoincidenceLogic;
188        part pressureLogic : CoincidenceLogic;
189        part saturationLogic : CoincidenceLogic;
190
191        part tempPressureTripOut : OrLogic;
192
193        connect temperatureLogic.actuate to tempPressureTripOut.channel1;
194        connect pressureLogic.actuate to tempPressureTripOut.channel2;
195       }
196       part def Actuator {(⊒ Actuator 2 (line 47 on page 89), Actuator 1 (line 45 on
              page 89), Actuator (line 275 on page 97), Actuators (line 43 on page 89))
197         // Actuate if either of these are true
198         port input: ActuationPort;
199         port manualActuatorInput: ~ActuationPort;
200        }
201     }
202
203     package EventControl {
204     import ScalarValues::Boolean;
205
206       port def MaintenancePort {
207         out maintenance: Boolean;
208       }
209
210       part def ControlUnit {
211         // Maintenance mode select x 4 instrumentation units
212         port maintenanceMode: MaintenancePort[4];
213         // Trip mode select x 4 instrumentation units
214         port tripMode: Instrumentation::TripModePort[4];
215         // New setpoints x 4 instrumentation units
216         port newPressureSetpoint: PressureSensor::PressureOutPort[4];
217         port newTemperatureSetpoint: TempSensor::TemperatureOutPort[4];
218         port newSaturationSetpoint: PressureSensor::PressureOutPort[4];
219         // Toggle actuator x2 actuators
220         port manualActuatorInput: Actuation::ActuationPort[2];
221       }
222     }
```

```
223 |
224 |   part RTS {(⊒ RTS (line 6 on page 85), RTS (line 94 on page 87))(⊑ RTS
        (line 36 on page 163))
225 |     part eventControl : EventControl::ControlUnit;
226 |
227 |     import Instrumentation::*;
228 |     part instrumentationAndSensing {
229 |       part pressureSensor1 : PressureSensor::'Pressure Sensor';
230 |       part pressureSensor2 : PressureSensor::'Pressure Sensor';
231 |
232 |       part tempSensor1 : TempSensor::'Temperature Sensor';
233 |       part tempSensor2 : TempSensor::'Temperature Sensor';
234 |
235 |       part instrumentationUnit1 : InstrumentationUnit;
236 |       part instrumentationUnit2 : InstrumentationUnit;
237 |       part instrumentationUnit3 : InstrumentationUnit;
238 |       part instrumentationUnit4 : InstrumentationUnit;
239 |
240 |       part tempDemux1 : TempSensor::Demux;
241 |       part tempDemux2 : TempSensor::Demux;
242 |
243 |       part pressureDemux1 : PressureSensor::Demux;
244 |       part pressureDemux2 : PressureSensor::Demux;
245 |
246 |       // Temp sensor 1
247 |       connect tempSensor1.output to tempDemux1.input;
248 |       connect tempDemux1.output1 to instrumentationUnit1.temperatureInput;
249 |       connect tempDemux1.output2 to instrumentationUnit2.temperatureInput;
250 |
251 |       // Temp sensor 2
252 |       connect tempSensor2.output to tempDemux2.input;
253 |       connect tempDemux2.output1 to instrumentationUnit3.temperatureInput;
254 |       connect tempDemux2.output2 to instrumentationUnit4.temperatureInput;
255 |
256 |       // Pressure sensor 1
257 |       connect pressureSensor1.output to pressureDemux1.input;
258 |       connect pressureDemux1.output1 to instrumentationUnit1.pressureInput;
259 |       connect pressureDemux1.output2 to instrumentationUnit2.pressureInput;
260 |
261 |       // Pressure sensor 2
262 |       connect pressureSensor2.output to pressureDemux2.input;
263 |       connect pressureDemux1.output1 to instrumentationUnit3.pressureInput;
264 |       connect pressureDemux1.output2 to instrumentationUnit4.pressureInput;
265 |     }
266 |
267 |     import Actuation::*;
268 |     part actuation {(⊑ Actuation (?? on page ??))
269 |       part actuationUnit1 : ActuationUnit;
270 |       part actuationUnit2 : ActuationUnit;
271 |
272 |       part actuator1 : Actuator;(⊒ Actuators (line 43 on page 89), Actuator
            (line 275 on page 97))
273 |       part actuator2 : Actuator;(⊒ Actuators (line 43 on page 89), Actuator
            (line 275 on page 97))
274 |
275 |       part actuateActuator1 : OrLogic;
276 |       part actuateActuator2 : OrLogic;
277 |
278 |       // connect actuators
279 |       // Actuator 1 ,temp or pressure trip
280 |       connect actuationUnit1.tempPressureTripOut.actuate to actuateActuator1.channel1;
281 |       connect actuationUnit2.tempPressureTripOut.actuate to actuateActuator1.channel2;
282 |       connect actuateActuator1.actuate to actuator1.input;
283 |
284 |       // Actuator 2 ,Saturation
285 |       connect actuationUnit1.saturationLogic.actuate to actuateActuator2.channel1;
286 |       connect actuationUnit2.saturationLogic.actuate to actuateActuator2.channel2;
```

```
287         connect actuateActuator2.actuate to actuator2.input;
288     }
289
290     // connect Control units
291     // Actuators manual override
292     connect eventControl.manualActuatorInput[1] to actuation.actuator1.manualActuatorInput;
293     connect eventControl.manualActuatorInput[2] to actuation.actuator2.manualActuatorInput;
294
295     // Instrumentation mode select
296     connect eventControl.maintenanceMode[1] to instrumentationAndSensing.
            ↪ instrumentationUnit1.setMaintenanceMode;
297     connect eventControl.maintenanceMode[2] to instrumentationAndSensing.
            ↪ instrumentationUnit2.setMaintenanceMode;
298     connect eventControl.maintenanceMode[3] to instrumentationAndSensing.
            ↪ instrumentationUnit3.setMaintenanceMode;
299     connect eventControl.maintenanceMode[4] to instrumentationAndSensing.
            ↪ instrumentationUnit4.setMaintenanceMode;
300
301     // Instrumentation pressure setpoint
302     connect eventControl.newPressureSetpoint[1] to instrumentationAndSensing.
            ↪ instrumentationUnit1.newPressureSetpoint;
303     connect eventControl.newPressureSetpoint[2] to instrumentationAndSensing.
            ↪ instrumentationUnit2.newPressureSetpoint;
304     connect eventControl.newPressureSetpoint[3] to instrumentationAndSensing.
            ↪ instrumentationUnit3.newPressureSetpoint;
305     connect eventControl.newPressureSetpoint[4] to instrumentationAndSensing.
            ↪ instrumentationUnit4.newPressureSetpoint;
306
307     // Instrumentation temperature setpoint
308     connect eventControl.newTemperatureSetpoint[1] to instrumentationAndSensing.
            ↪ instrumentationUnit1.newTemperatureSetpoint;
309     connect eventControl.newTemperatureSetpoint[2] to instrumentationAndSensing.
            ↪ instrumentationUnit2.newTemperatureSetpoint;
310     connect eventControl.newTemperatureSetpoint[3] to instrumentationAndSensing.
            ↪ instrumentationUnit3.newTemperatureSetpoint;
311     connect eventControl.newTemperatureSetpoint[4] to instrumentationAndSensing.
            ↪ instrumentationUnit4.newTemperatureSetpoint;
312
313     // Instrumentation saturation setpoint
314     connect eventControl.newSaturationSetpoint[1] to instrumentationAndSensing.
            ↪ instrumentationUnit1.newSaturationSetpoint;
315     connect eventControl.newSaturationSetpoint[2] to instrumentationAndSensing.
            ↪ instrumentationUnit2.newSaturationSetpoint;
316     connect eventControl.newSaturationSetpoint[3] to instrumentationAndSensing.
            ↪ instrumentationUnit3.newSaturationSetpoint;
317     connect eventControl.newSaturationSetpoint[4] to instrumentationAndSensing.
            ↪ instrumentationUnit4.newSaturationSetpoint;
318
319     // Instrumentation trip mode
320     // Bypass temperature
321     connect eventControl.tripMode[1] to instrumentationAndSensing.instrumentationUnit1.
            ↪ tripMode;
322     connect eventControl.tripMode[2] to instrumentationAndSensing.instrumentationUnit2.
            ↪ tripMode;
323     connect eventControl.tripMode[3] to instrumentationAndSensing.instrumentationUnit3.
            ↪ tripMode;
324     connect eventControl.tripMode[4] to instrumentationAndSensing.instrumentationUnit4.
            ↪ tripMode;
325
326     // Trip on pressure above the setpoint
327     // Actuation unit 1
328     connect instrumentationAndSensing.instrumentationUnit1.pressureTripOut to actuation.
            ↪ actuationUnit1.pressureLogic.channel1;
329     connect instrumentationAndSensing.instrumentationUnit2.pressureTripOut to actuation.
            ↪ actuationUnit1.pressureLogic.channel2;
330     connect instrumentationAndSensing.instrumentationUnit3.pressureTripOut to actuation.
            ↪ actuationUnit1.pressureLogic.channel3;
```

```
331       connect instrumentationAndSensing.instrumentationUnit4.pressureTripOut to actuation.
              ↪ actuationUnit1.pressureLogic.channel4;
332       // Actuation unit 2
333       connect instrumentationAndSensing.instrumentationUnit1.pressureTripOut to actuation.
              ↪ actuationUnit2.pressureLogic.channel1;
334       connect instrumentationAndSensing.instrumentationUnit2.pressureTripOut to actuation.
              ↪ actuationUnit2.pressureLogic.channel2;
335       connect instrumentationAndSensing.instrumentationUnit3.pressureTripOut to actuation.
              ↪ actuationUnit2.pressureLogic.channel3;
336       connect instrumentationAndSensing.instrumentationUnit4.pressureTripOut to actuation.
              ↪ actuationUnit2.pressureLogic.channel4;
337
338       // Trip on temperature above the setpoint
339       // Actuation unit 1
340       connect instrumentationAndSensing.instrumentationUnit1.temperatureTripOut to actuation.
              ↪ actuationUnit1.temperatureLogic.channel1;
341       connect instrumentationAndSensing.instrumentationUnit2.temperatureTripOut to actuation.
              ↪ actuationUnit1.temperatureLogic.channel2;
342       connect instrumentationAndSensing.instrumentationUnit3.temperatureTripOut to actuation.
              ↪ actuationUnit1.temperatureLogic.channel3;
343       connect instrumentationAndSensing.instrumentationUnit4.temperatureTripOut to actuation.
              ↪ actuationUnit1.temperatureLogic.channel4;
344       // Actuation unit 2
345       connect instrumentationAndSensing.instrumentationUnit1.temperatureTripOut to actuation.
              ↪ actuationUnit2.temperatureLogic.channel1;
346       connect instrumentationAndSensing.instrumentationUnit2.temperatureTripOut to actuation.
              ↪ actuationUnit2.temperatureLogic.channel2;
347       connect instrumentationAndSensing.instrumentationUnit3.temperatureTripOut to actuation.
              ↪ actuationUnit2.temperatureLogic.channel3;
348       connect instrumentationAndSensing.instrumentationUnit4.temperatureTripOut to actuation.
              ↪ actuationUnit2.temperatureLogic.channel4;
349
350       // Trip on saturation above the setpoint
351       // Actuation unit 1
352       connect instrumentationAndSensing.instrumentationUnit1.saturationTripOut to actuation.
              ↪ actuationUnit1.saturationLogic.channel1;
353       connect instrumentationAndSensing.instrumentationUnit2.saturationTripOut to actuation.
              ↪ actuationUnit1.saturationLogic.channel2;
354       connect instrumentationAndSensing.instrumentationUnit3.saturationTripOut to actuation.
              ↪ actuationUnit1.saturationLogic.channel3;
355       connect instrumentationAndSensing.instrumentationUnit4.saturationTripOut to actuation.
              ↪ actuationUnit1.saturationLogic.channel4;
356       // Actuation unit 2
357       connect instrumentationAndSensing.instrumentationUnit1.saturationTripOut to actuation.
              ↪ actuationUnit2.saturationLogic.channel1;
358       connect instrumentationAndSensing.instrumentationUnit2.saturationTripOut to actuation.
              ↪ actuationUnit2.saturationLogic.channel2;
359       connect instrumentationAndSensing.instrumentationUnit3.saturationTripOut to actuation.
              ↪ actuationUnit2.saturationLogic.channel3;
360       connect instrumentationAndSensing.instrumentationUnit4.saturationTripOut to actuation.
              ↪ actuationUnit2.saturationLogic.channel4;
361
362     }
363   } // package id RTS_System_Arch 'RTS System Architecture'
364 } // package id Architecture 'RTS Architecture'
```

D.14 RTS Viewpoints

Listing D.14: Listing SysML Model of RTS_Viewpoints.

```
1 import Views::*;
2
3 package 'RTS Viewpoints and Views' {
```

```
  import 'RTS Stakeholders'::*;

  // Viewpoints and view definitions specify the different points of view
  // that a system may be viewed from, and which parts of the system model
  // attend to each view. We will specify viewpoints for a few kinds
  // of NRC actors, various kinds of HARDENS performers, various kinds of
  // other Galois employees who are interested in this project and its
  // outcomes, and other government and industry parties interested in RDE.

  viewpoint 'NRC General Customer Viewpoint' {}

  view def 'NRC General Customer View Definition' {}

  viewpoint 'NRC Assurance Customer Viewpoint' {}

  view def 'NRC Assurance Customer View Definition' {}

  viewpoint 'Galois Performer Viewpoint' {}

  view def 'Galois Performer View Definition' {}

  view 'Galois PI View' : 'Galois Performer View Definition' {}

  view 'Galois PL View' : 'Galois Performer View Definition' {}

  view 'Galois Software Engineer View' : 'Galois Performer View Definition' {}

  view 'Galois Hardware Engineer View' : 'Galois Performer View Definition' {}

  view 'Galois Assurance Engineer View' : 'Galois Performer View Definition' {}

  viewpoint 'Galois Principal View' {}

  viewpoint 'Galois Executive View' {}

  viewpoint 'Galois Customer Specialist View' {}

  viewpoint 'Galois Research Engineering View' {}

  viewpoint 'General Party interested in Rigorous Digital Engineering' {}
}
```

D.15 Semantic Properties

Listing D.15: SysMLv2 Model of Semantic Properties.

```
/** Semantic properties are annotation to model and system artifacts
    used to semantically markup those artifacts for documentation,
    traceability, and more. */
package 'Semantic Properties' {
  doc /* Semantic Properties are used to document arbitrary
         constructs in our specifications and implementations.
         @see https://www.kindsoftware.com/documents/whitepapers/code_standards/properties.
              ↪ html
       */
  import ScalarValues::*;

  attribute def id SP 'Semantic Property';
  attribute def id SPD 'Semantic Property with Description' :>
    'Semantic Property' {
    attribute description: String;
  }
```

```
attribute def Exception :> String;

package 'Meta,Information' {
  attribute def Author :> SP {
    author: String;
  }
  attribute def Lando :> SP {
    summary: String;
  }
  attribute def Bug :> SPD;
  attribute def Copyright :> SP {
    copyright: String;
  }
  attribute def Description :> SPD;
  attribute def History :> SPD;
  attribute def License :> SP {
    license: String;
  }
  attribute def Title :> SP {
    title: String;
  }
}

attribute def 'Author Description Scope Triple' :> SPD {
  import 'Meta,Information'::*;
  attribute author: Author;
  attribute scope: Boolean;
}
package 'Pending Work' {
  attribute def Idea :> 'Author Description Scope Triple' {
    classifier: String;
  }
  attribute def Review :> 'Author Description Scope Triple';
  attribute def Todo :> 'Author Description Scope Triple';
}

attribute def 'Rich Assertion' :> SPD {
  attribute label: String;
  attribute expression: Boolean;
  attribute exception : Exception;
}

attribute def 'Expression Description Pair' :> SPD {
  attribute expression: Boolean;
}

package 'Formal Specifications' {
  import Collections::*;
  enum def 'Modifies Frame' {
    SINGLE_ASSIGNMENT;
    QUERY;
    EXPRESSION;
  }
  attribute def Ensures :> 'Rich Assertion';
  attribute def Generate :> 'Expression Description Pair';
  attribute def Invariant :> 'Expression Description Pair' {
    exception: Exception;
  }
  attribute def Modify :> 'Expression Description Pair' {
    enum kind: 'Modifies Frame';
  }
  attribute def requires :> 'Rich Assertion';
}

package 'Concurrency Control' {
  import Collections::*;
```

```
    attribute def Locks :> Set;
    attribute def Timeout {
      attribute timeout: Natural;
      attribute exception: String;
    }
    attribute def 'Concurrency Semantic Property' :> 'Semantic Property' {
      attribute locks: Locks;
      attribute failure: Exception;
      attribute atomic: Boolean;
      attribute special: String;
      attribute timeout: Timeout;
    }
    attribute def Concurrent :> 'Concurrency Semantic Property' {
      attribute threadcountlimit: Positive;
      attribute broken: Boolean;
    }
    attribute def Sequential :> 'Concurrency Semantic Property';
    attribute def Guarded :> 'Concurrency Semantic Property' {
      attribute semaphore_count: Positive;
    }
  }

  package 'Usage Information' {
    attribute def 'Parameter Spec' {
      parameter_name: String;
      precondition: Boolean;
      description: String;
    }
    attribute def Return :> String;
    attribute def Exception :> 'Expression Description Pair' {
      exception: Exception;
    }
  }

  package Versioning {
    attribute def Version :> String;
    attribute def Deprecated :> String;
    attribute def Since :> String;
  }

  attribute def 'Feature Name Description Pair' {
    feature_name: String;
    description: String;
  }
  package Inheritance {
    attribute def Hides :> 'Feature Name Description Pair';
    attribute def Overrides :> 'Feature Name Description Pair';
  }

  package Documentation {
    attribute def Design :> 'Author Description Scope Triple';
    attribute def Equivalent :> String;
    attribute def Example :> String;
    attribute def See :> String;
  }

  package Dependencies {
    import Collections::*;
    attribute def References :> 'Expression Description Pair';
    // Note that we rename 'use' to 'uses' to avoid SysML keyword conflict.
    attribute def Uses :> 'Expression Description Pair';
  }

  package Miscellaneous {
    attribute def Guard :> 'Expression Description Pair';
    attribute def Values :> 'Expression Description Pair';
```

```
    attribute def 'Time Complexity' :> 'Expression Description Pair';
    attribute def 'Space Complexity' :> 'Expression Description Pair';
  }
}
```

Appendix E

Cryptol Model

E.1 Top-level RTS Cryptol model

Listing E.1: Listing Cryptol Model of RTS.

```
1  // HARDENS Reactor Trip System (RTS)
2  // A formal model of RTS behavior written in the Cryptol DSL.
3  //
4  // @author Alex Bakst <abakst@galois.com>
5  // @created November, 2021
6  // @refines HARDENS.sysml
7  // @refines RTS.lando
8  // @refines RTS_Requirements.json
9
10 module RTS where
11
12 import RTS::Utils
13 import RTS::InstrumentationUnit
14 import RTS::InstrumentationUnit as I
15 import RTS::ActuationUnit
16 import RTS::ActuationUnit as AU
17 import RTS::Actuator
18 import RTS::Actuator as A
19
20 type Device = [1](⊒  Device   (?? on page ??))
21 type Division = [2]
22 type Logic = [1]
23
24 D0, D1: Device
25 D0 = 0
26 D1 = 1
27
28 type EventType = [2]
29 // Event Type Values
30 EVENT_MAINTENANCE, EVENT_TRIP_MODE, EVENT_SETPOINT, EVENT_ACTUATE : EventType
31 EVENT_MAINTENANCE = 0
32 EVENT_TRIP_MODE = 1
33 EVENT_SETPOINT = 2
34 EVENT_ACTUATE = 3
35
36 type RTS =(⊒  RTS   (?? on page ??),   RTS   (?? on page ??),   RTS   (line 224 on
       page 156))
37   { control: Control
38   , instrumentation: Instrumentation
```

```
  , actuation: Actuation
  , ui: UI
  }

type UI =
  { instrumentation_values: [4][NChannels][32]
  , instrumentation_bypass: [4][NChannels]
  , instrumentation_trip: [4][NChannels]
  }

type Control =
  { self_test: Bit // Are we running an end-to-end test?
  , self_test_instrs: [2]Division // Which instrumentation units are we poking?
  , self_test_channel: Channel // Which channel are we poking?
  , self_test_logic: Logic // Which actuation logic unit are we observing?
  , self_test_dev: Device // Which actuator are we observing?
  , self_test_timer: [32]
  , self_test_fail: Bit
  }

type Actuation =
  { units: [2]ActuationUnit
  , actuators: [2]Actuator
  }

type Instrumentation =
  { units: [4]InstrumentationUnit }

type Event =
  { event: EventType // The event tag
  , ch: Channel // Which channel
  , dev: Device // Which device
  , unit: Division // Which Instrumentation unit
  , setpoint: [32]
  , mode: I::Mode
  , on_off: Bit
  }

// @refines ControlUnit
Event_Control: Event -> RTS -> RTS
Event_Control e rts =
  if e.event == EVENT_MAINTENANCE then
    on_instr_units rts (\units -> update units (e.unit) (Set_Maintenance e.on_off (units @ e.
        ↪ unit)))
  else if (e.event == EVENT_TRIP_MODE) && (ch < `NChannels) && (e.mode < `NModes) then
    on_instr_units rts (\units -> update units (e.unit) (Set_Mode ch e.mode (units @ e.unit))
        ↪ )
  else if (e.event == EVENT_SETPOINT) && (ch < `NChannels) then
    on_instr_units rts (\units -> update units (e.unit) (Set_Setpoint ch e.setpoint (units @
        ↪ e.unit)))
  else if (e.event == EVENT_ACTUATE) then
    {rts| actuation =
      {rts.actuation | actuators = update rts.actuation.actuators dev (SetManual e.on_off (
          ↪ rts.actuation.actuators @ dev))} }
  else rts
  where
    ch : Channel
    ch = e.ch
    dev : Device
    dev = e.dev

    on_instr_units r f = {r|instrumentation = {units = f r.instrumentation.units }}

Sense_Actuate:
  [4]I::Input ->
  [4]I::Command ->
```

```
  RTS ->
  RTS
Sense_Actuate inputs icmds rts =
  { rts | control = { rts.control | self_test_fail = test_fail }
        , instrumentation = { units = instrumentation' }
        , actuation = { units = actuation_logic', actuators = actuators' }
        }
  where
    // The two instrumentation units under test
    i0 = rts.control.self_test_instrs @ 0
    i1 = rts.control.self_test_instrs @ 1
    test_ch = rts.control.self_test_channel
    test_dev = rts.control.self_test_dev
    test_logic = rts.control.self_test_logic

    // Each instrumentation unit runs
    instrumentation' = [ I::Step input cmd i | i <- rts.instrumentation.units | input <-
        ↪ inputs | cmd <- icmds ]
    output_trips: [3][4]TripPort
    output_trips = [[ instr.output_trip @ ch | instr <- instrumentation' ] | ch <- [0..2]]

    // Mask trips from the instrs under test at the channel under test. This is passed
    // to the voting components _not_ under test (so that we do not count any test-generated
        ↪ trips)
    output_trips_masked, output_trips_test: [3][4]TripPort
    output_trips_masked = MaskTripFrom (\i ch -> elem i [i0, i1] && (ch == test_ch))
        ↪ output_trips
    output_trips_test = MaskTripFrom (\i ch -> ~ (elem i [i0, i1] && (ch == test_ch)))
        ↪ output_trips

    // Now run the voting logic
    actuation_logic' : [2]ActuationUnit
    actuation_logic' = [ AU::TripInput (Select_trips l) logic | logic <- rts.actuation.units
        ↪ | l <- [0..1] ]

    // Mask out the output of the voting unit we are *not* testing, and then use *this* value
    // for testing device actuation
    test_actuation_logic : [2]ActuationUnit
    test_actuation_logic = [ { output = [ if (test_logic == l) && (test_dev == d) then logic.
        ↪ output @ d else 0 | d <- [0..1] ]}
                           | logic <- actuation_logic' | l <- [0..1]
                         ]

    Select_trips l = if rts.control.self_test then
                       (if l == test_logic then output_trips_test else output_trips_masked)
                     else
                       output_trips

    // This is the output to the devices
    actuators' : [2]Actuator
    actuators' = [ A::SetInput (A::ActuateActuator [ logic.output @ d | logic <-
        ↪ actuation_logic' ]) a | a <- rts.actuation.actuators | d <- [0...] ]

    actuators_test : [2]Actuator
    actuators_test = [ A::SetInput (A::ActuateActuator [ logic.output @ d | logic <-
        ↪ test_actuation_logic ]) a | a <- rts.actuation.actuators | d <- [0...] ]

    old_vote = (rts.actuation.units @ test_logic).output @ test_dev
    expect = (old_vote != 0) || ShouldActuate test_ch inputs rts.instrumentation.units [i0,i1
        ↪ ]
    test_fail = expect != (actuators_test @ test_dev).input

// @review kiniry Shouldn't this/these be private?
MaskTripFrom: {idx} (Integral idx, Literal 0 idx) =>
  (idx -> Channel -> Bit) -> [3][4][8] -> [3][4][8]
MaskTripFrom p trips =
  [ [ if p j ch then 0 else ch_trip | ch_trip <- trip | j <- [0...] ]
```

```
  | trip <- trips
  | ch <- [0..2]
  ]

Test_Instrumentation : {n} (fin n) =>
  [n]([8], [2][32], Channel) -> [2] -> RTS -> RTS
Test_Instrumentation tests i rts = {rts | control = {rts.control|self_test_fail = ~ (all
    ↪ Run_test tests)}}
  where
    Run_test: ([8],[2][32],Channel) -> Bit
    Run_test (trip,vals,ch) = (rts.instrumentation.units @ i)
                      >>>> Set_Maintenance True
                      >>>> Set_Mode ch Operate
                      >>>> Set_Maintenance False
                      >>>> (\in -> I::Step vals I::NoCommand in)
                      >>>> (\in -> trip == (in.output_trip @ ch))

Test_Voting: {n} (fin n) => [n](Bit, [1], AU::Input) -> [1] -> RTS -> RTS
Test_Voting tests l rts = {rts | control = {rts.control|self_test_fail = ~ (all Run_test
    ↪ tests)}}
  where
    Run_test (on,d,trips) = (rts.actuation.units @ l)
                    >>>> AU::TripInput trips
                    >>>> \a -> (a.output @ d) == on

SelfTestOracle: [4]I::Input -> [4][3][32] -> [2][2] -> [2]
SelfTestOracle inputs setpoints [i0,i1] = [(ts == 0b11) || (ps == 0b11), ss == 0b11]
  where
     ts,ps,ss:[2]
     ts = [ValueShouldTrip T i s | i <- inputs @@ [i0,i1] | s <- setpoints @@ [i0,i1]]
     ps = [ValueShouldTrip P i s | i <- inputs @@ [i0,i1] | s <- setpoints @@ [i0,i1]]
     ss = [ValueShouldTrip S i s | i <- inputs @@ [i0,i1] | s <- setpoints @@ [i0,i1]]

// Helpful for testcase generation. The `off` parameter is used
// to generate a second instrumentation division ID that's guaranteed
// to be different from the first one (i1)
SelfTestOracleHalf: [4][2][16] -> [4][3][16] -> [2] -> [2] -> [2]
SelfTestOracleHalf inputs setpoints i1 off =
  SelfTestOracle (map (map extend) inputs) (map (map extend) setpoints) [i1, i2]
  where
    i2 = i1 + 1 + (off % 3)
    extend v = zero # v

private
    property sense_to_actuate_0 rts (sensors : [4][2][32]) cmds =
        ~rts.control.self_test ==> (
           sum [ if Is_Tripped T i then 1 else 0 | i <- rts'.instrumentation.units ] >= 2
        \/ sum [ if Is_Tripped P i then 1 else 0 | i <- rts'.instrumentation.units ] >= 2
        \/ or [ a.output @ D0 | a <- rts.actuation.units ]
        /* -------------------- */ ==
        (rts'.actuation.actuators @ D0).input
      )
      where
        rts' = rts >>>> Sense_Actuate sensors cmds

    property sense_to_actuate_1 rts (sensors : [4][2][32]) cmds =
        ~rts.control.self_test ==> (
           sum [ if Is_Tripped S i then 1 else 0 | i <- rts'.instrumentation.units ] >= 2
        \/ or [ a.output @ D1 | a <- rts.actuation.units ]
        /* -------------------- */ ==
        (rts'.actuation.actuators @ D1).input
      )
      where
        rts' = rts >>>> Sense_Actuate sensors cmds

    property end_to_end_test (ch : [2]) rts i1 i2 vote sensors =
```

```
      ( ~rts.control.self_test_fail
        ∧ (i1 != i2)
        ∧ and [ (a.output == 0) | a <- rts.actuation.units ]
        ∧ (ch < 3 /* not provable if ch == S */))// ∧ (ch != S) // Add this to make it
            ↪ provable
      /* ----------------------------------------*/ ==>
        ~rts'.control.self_test_fail
    where
      rts' = RunEndToEnd rts i1 i2 ch vote (repeat sensors)

property end_to_end_test_non_interference (ch : [2]) rts (sensors: [4][2][32]) =
  (~ rts.control.self_test_fail
    // Start in a state where we haven't actuated
    ∧ and [ (a.output == 0) | a <- rts.actuation.units ]
    ∧ ch < 3
    // Check if enough of the non-tested instrumentation (which may be all of them!) thinks
        ↪ we should
    // actuate
    ∧ sum [ if (~rts.control.self_test || ~elem i [i1,i2]) && ChShouldTrip ch vs instr
        ↪ then 1 else 0
          | vs <- sensors
          | instr <- rts.instrumentation.units
          | i <- [0...]
          ] >= 2
  ) /* ------------------------------------------------------------ */ ==>
  (rts'.actuation.actuators @ (if ch == S then D1 else D0)).input
  where
    i1 = rts.control.self_test_instrs@0
    i2 = rts.control.self_test_instrs@1
    rts' = rts >>>> Sense_Actuate sensors (repeat I::NoCommand)

property test_instrumentation_ok i rts vs (ch : Channel) =
  ~ (rts.control.self_test_fail) ==> (
      (ch < 3) ==>
      (~(Test_Instrumentation [(pass, vs, ch)] i rts).control.self_test_fail)
    )
  where
    pass: [8]
    pass = if ChShouldTrip ch vs {(rts.instrumentation.units @ i) | mode = repeat Operate
        ↪ } then 1 else 0

property test_voting_ok d ts l rts =
  (~rts.control.self_test_fail && and [ a.output == 0 | a <- rts.actuation.units ]) ==>
  ~ ((Test_Voting [(expect, d, ts)] l rts).control.self_test_fail)
  where
    expect =
      if d == D0 then
          (sum [ if (t != 0) then 1 else 0 | t <- ts @ T ] >= 2)
        || (sum [ if (t != 0) then 1 else 0 | t <- ts @ P ] >= 2)
      else
          sum [ if (t != 0) then 1 else 0 | t <- ts @ S ] >= 2

I : ([4]InstrumentationUnit -> [4]InstrumentationUnit) -> RTS -> RTS
I f rts = {rts|instrumentation = { units = f rts.instrumentation.units }}

RunEndToEnd rts i1 i2 ch vote sensors = rts'
  where
    mkTestCore : [4]InstrumentationUnit -> Control
    mkTestCore instrs = {rts.control| self_test = True,
                                      self_test_instrs = [i1, i2],
                                      self_test_channel = ch,
                                      self_test_logic = vote,
                                      self_test_dev = dev }
    dev = if (ch == T) || (ch == P) then D0 else D1
    rts' = rts
```

```
        >>>> I (\instrs -> [ if (i == i1) || (i == i2) then Set_Maintenance True instr else
             ↪ instr | instr <- instrs | i <- [0..3] ])
        >>>> I (\instrs -> [ if (i == i1) || (i == i2) then Set_Mode ch Operate instr else
             ↪ instr | instr <- instrs | i <- [0..3] ])
        >>>> (\r -> {r | control = (mkTestCore r.instrumentation.units) })
        >>>> Sense_Actuate sensors cmds
      cmds = repeat I::NoCommand

  ShouldActuate: Channel -> [4][2][32] -> [4]InstrumentationUnit -> [2][2] -> Bit
  ShouldActuate ch sensors instrumentation test_instrs =
    numBits [ ChShouldTrip ch vs instr
            | vs <- sensors @@ test_instrs | instr <- instrumentation @@ test_instrs ] >= 2

  ChShouldTrip ch vs instr =
    if In_Mode ch Operate instr then
      ValueShouldTrip ch vs (instr.setpoints)
    else
      In_Mode ch Manual instr

  ValueShouldTrip ch vs sp =
      if (ch == T) || (ch == P)
      then (sp @ ch) < (vs @ ch)
      else (sp @ ch) >$ Saturation (vs @ T) (vs @ P)
```

E.2 Cryptol model of the Actuation Unit

Listing E.2: Listing Cryptol Model of ActuationUnit.

```
// HARDENS Reactor Trip System (RTS) Actuation Unit
// A formal model of RTS Actuation Unit behavior written in the
// Cryptol DSL.
//
// @author Alex Bakst <abakst@galois.com>
// @created November, 2021
// @refines HARDENS.sysml
// @refines RTS.lando
// @refines RTS_Requirements.json

module RTS::ActuationUnit where

import RTS::Utils
import RTS::InstrumentationUnit

type Input = [3][4]TripPort
type ActuationPort = Bit
type ActuationUnit = { output: [2]ActuationPort }
type CoincidenceLogic = [4]TripPort -> Bit
type OrLogic = [2] -> Bit

TripInput: [3][4]TripPort -> ActuationUnit -> ActuationUnit
TripInput trips actuationUnit =
  { output = [ d0, d1 ] }
  where
    d0 = Actuate_D0 trips (actuationUnit.output @ 0)
    d1 = Actuate_D1 trips (actuationUnit.output @ 1)

Actuate_D0: Input -> Bit -> Bit
Actuate_D0 trips old = d0 || old
  where
    d0 = TempPressureTripOut [TemperatureLogic temperatureTrips,
                              PressureLogic pressureTrips]
    temperatureTrips = trips @ (0:[2])
```

```
35 |    pressureTrips = trips @ (1:[2])
36 |
37 | Actuate_D1: Input -> Bit -> Bit
38 | Actuate_D1 trips old = d1 || old
39 |   where
40 |     d1 = SaturationLogic saturationTrips
41 |     saturationTrips = trips @ (2:[2])
42 |
43 | TemperatureLogic: [4]TripPort -> Bit
44 | TemperatureLogic ts = Coincidence_2_4 ts
45 |
46 | PressureLogic: [4]TripPort -> Bit
47 | PressureLogic ts = Coincidence_2_4 ts
48 |
49 | SaturationLogic: [4]TripPort -> Bit
50 | SaturationLogic ts = Coincidence_2_4 ts
51 |
52 | TempPressureTripOut: [2] -> Bit
53 | TempPressureTripOut ts = (ts @ (0:[1])) || (ts @ (1:[1]))
54 |
55 | private
56 |   Coincidence_2_4 : [4]TripPort -> Bit
57 |   Coincidence_2_4 x =
58 |     (a&&b) || ((a||b) && (c||d)) || (c&&d)
59 |     where
60 |       a = (x @ (0:[2])) != 0
61 |       b = (x @ (1:[2])) != 0
62 |       c = (x @ (2:[2])) != 0
63 |       d = (x @ (3:[2])) != 0
64 |
65 |   count : {n} (fin n, n >= 1) => [n] -> [width n]
66 |   count bs = sum [ if b then 1 else 0 | b <- bs ]
67 |
68 |   /** @requirements
69 |         ACTUATION_LOGIC_VOTE_TEMPERATURE
70 |   */
71 |   property actuation_logic_vote_temperature (inp: Input) =
72 |     (count [i != 0 | i <- (inp @ T)] >= 2) == TemperatureLogic (inp @ T)
73 |
74 |   /** @requirements
75 |         ACTUATION_LOGIC_VOTE_PRESSURE
76 |   */
77 |   property actuation_logic_vote_pressure (inp: Input) =
78 |     (count [i != 0 | i <- (inp @ P)] >= 2) == PressureLogic (inp @ P)
79 |
80 |   /** @requirements
81 |         ACTUATION_LOGIC_VOTE_SATURATION
82 |   */
83 |   property actuation_logic_vote_saturation (inp: Input) =
84 |     (count [i != 0 | i <- (inp @ S)] >= 2) == SaturationLogic (inp @ S)
85 |
86 |   /** @requirements
87 |         ACTUATION_LOGIC_DEVICE_0
88 |         ACTUATION_LOGIC_VOTE_DEVICE_0
89 |         ACTUATION_LOGIC_VOTE_MANUAL_DEVICE_0
90 |   */
91 |   property actuation_logic_device_0 (inp: Input) (act: ActuationUnit) =
92 |       act.output @ 0
93 |     \/ TempPressureTripOut [TemperatureLogic (inp @ T), PressureLogic (inp @ P)]
94 |     == /* ---------------- */
95 |     act'.output @ 0
96 |     where act' = TripInput inp act
97 |
98 |   /** @requirements
99 |         ACTUATION_LOGIC_DEVICE_1
100|         ACTUATION_LOGIC_VOTE_DEVICE_1
```

```
      ACTUATION_LOGIC_VOTE_MANUAL_DEVICE_1
 */
 property actuation_logic_device_1 (inp: Input) (act: ActuationUnit) =
     act.output @ 1
   ∨ SaturationLogic (inp @ S)
   == /* ---------------- */
   act'.output @ 1
   where act' = TripInput inp act
```

E.3 Cryptol model of the Actuator

Listing E.3: Listing Cryptol Model of Actuator.

```
// HARDENS Reactor Trip System (RTS) Actuator Unit
// A formal model of RTS Actuator behavior written in the Cryptol
// DSL.
//
// @author Alex Bakst <abakst@galois.com>
// @created November, 2021
// @refines HARDENS.sysml
// @refines RTS.lando
// @refines RTS_Requirements.json

module RTS::Actuator where

type Actuation = Bit
type Mode = Bit

/** @requirements
     ACTUATION_LOGIC_MANUAL_DEVICE_{0,1} satisfied by definition
*/
type Actuator =
  { input: Actuation
  , manualActuatorInput: Actuation
  }

SetInput: Actuation -> Actuator -> Actuator
SetInput on actuator = {actuator | input = on }

SetManual: Actuation -> Actuator -> Actuator
SetManual on actuator = {actuator | manualActuatorInput = on}

ActuateActuator : [2]Actuation -> Actuation
ActuateActuator inputs = (inputs @ (0:[1])) || (inputs @ (1:[1]))
```

E.4 Cryptol model of the Instrumentation Unit

Listing E.4: Listing Cryptol Model of InstrumentationUnit.

```
// HARDENS Reactor Trip System (RTS) Instrumentation Unit
// A formal model of RTS Instrumentation behavior written in the
// Cryptol DSL.
//
// @author Alex Bakst <abakst@galois.com>
// @created November, 2021
// @refines HARDENS.sysml
// @refines RTS.lando
// @refines RTS_Requirements.json
```

```
10 |
11 | module RTS::InstrumentationUnit where
12 |
13 | import RTS::Utils
14 |
15 | type Input = [2][32]
16 | type NChannels = 3
17 | type Channel = [lg2 NChannels]
18 | type TripPort = [8]
19 | type NModes = 3
20 | type Mode = [lg2 NModes]
21 | type CommandType = [lg2 NCommands]
22 | type NCommands = 4
23 | type Command =
24 |   { command: CommandType
25 |   , channel: Channel
26 |   , mode: Mode
27 |   , setpoint: [32]
28 |   , on_off: Bit
29 |   }
30 |
31 | Set_Mode_Cmd, Set_Maintenance_Cmd, Set_Setpoint_Cmd, Null_Cmd : CommandType
32 | Set_Mode_Cmd = 0
33 | Set_Maintenance_Cmd = 1
34 | Set_Setpoint_Cmd = 2
35 | Null_Cmd = 3
36 |
37 | T,P,S : Channel
38 | T = 0 // Temperature
39 | P = 1 // Pressure
40 | S = 2 // Saturation
41 |
42 | Bypass,Operate,Manual: Mode
43 | Bypass = 0 // Do not generate a "trip"
44 | Operate = 1 // Generate "trip" on setpoint violation
45 | Manual = 2 // Force "trip" generation
46 |
47 | type InstrumentationUnit =
48 |   { setpoints: [NChannels][32]
49 |   , reading: [NChannels][32]
50 |   , mode: [NChannels]Mode
51 |   , sensor_trip: [NChannels]
52 |   , output_trip: [NChannels][8]
53 |   , maintenance: Bit
54 |   }
55 |
56 | NoCommand: Command
57 | NoCommand = { command= Null_Cmd, channel= zero, mode= zero, setpoint= zero, on_off= zero }
58 |
59 | Initial: InstrumentationUnit
60 | Initial =
61 |   { setpoints = zero
62 |   , reading = zero
63 |   , mode = zero
64 |   , sensor_trip = zero
65 |   , output_trip = repeat zero
66 |   , maintenance = ~zero
67 |   }
68 |
69 | Step: Input -> Command -> InstrumentationUnit -> InstrumentationUnit
70 | Step inp cmd instr =
71 |       Handle_Input inp instr
72 |   >>>> Handle_Command cmd
73 |   >>>> Step_Trip_Signals
74 |   >>>> OutputTrips
75 |   where
```

```
76  |    OutputTrips state =
77  |      { state | output_trip = [ zero # [Is_Ch_Tripped (state.mode @ ch) (state.sensor_trip @
    |           ↪ ch)] | ch <- [0..2] ] }
78  |
79  | // @refines sensor input ports
80  | Handle_Input: Input -> InstrumentationUnit -> InstrumentationUnit
81  | Handle_Input sensors instr = { instr | reading = vals }
82  |   where
83  |     vals: [3][32]
84  |     // 0 and 1 should be T and P, but this causes problems for crymp
85  |     vals = [ sensors @ 0
86  |            , sensors @ 1
87  |            , Saturation (sensors @ 0) (sensors @ 1)
88  |            ]
89  |
90  | // @refines mode, tripmode, setpoint input port attributes
91  | Handle_Command: Command -> InstrumentationUnit -> InstrumentationUnit
92  | Handle_Command cmd instr =
93  |   if (cmd.command == Set_Mode_Cmd) && (cmd.channel < `NChannels) then
94  |     Set_Mode cmd.channel cmd.mode instr
95  |   else if cmd.command == Set_Maintenance_Cmd then
96  |     Set_Maintenance cmd.on_off instr
97  |   else if (cmd.command == Set_Setpoint_Cmd) && (cmd.channel < `NChannels) then
98  |     Set_Setpoint cmd.channel cmd.setpoint instr
99  |   else
100 |     instr
101 |
102 | ////////////////////////////////////////
103 | // Queries, "Setters", "Getters", etc
104 | ////////////////////////////////////////
105 |
106 | Get_Reading: InstrumentationUnit -> [NChannels][32]
107 | Get_Reading instr = instr.reading
108 |
109 | In_Maintenance: InstrumentationUnit -> Bit
110 | In_Maintenance instr = instr.maintenance
111 |
112 | Set_Maintenance: Bit -> InstrumentationUnit -> InstrumentationUnit
113 | Set_Maintenance on instr = { instr | maintenance = on }
114 |
115 | Set_Mode: Channel -> Mode -> InstrumentationUnit -> InstrumentationUnit
116 | Set_Mode ch mode i =
117 |   if In_Maintenance i && (mode <= 2) then
118 |     {i | mode = update i.mode ch mode}
119 |   else
120 |     i
121 |
122 | In_Mode: Channel -> Mode -> InstrumentationUnit -> Bit
123 | In_Mode ch mode instr = (instr.mode @ ch) == mode
124 |
125 | Get_Setpoint: Channel -> InstrumentationUnit -> [32]
126 | Get_Setpoint ch instr = instr.setpoints @ ch
127 |
128 | Set_Setpoint: Channel -> [32] -> InstrumentationUnit -> InstrumentationUnit
129 | Set_Setpoint ch val instr = { instr | setpoints = update instr.setpoints ch val }
130 |
131 | Get_Tripped: InstrumentationUnit -> [NChannels][8]
132 | Get_Tripped instr = [zero # [Is_Tripped T instr], zero # [Is_Tripped P instr], zero # [
    |      ↪ Is_Tripped S instr]]
133 |
134 | Is_Tripped: Channel -> InstrumentationUnit -> Bit
135 | Is_Tripped ch instr = In_Mode ch Manual instr
136 |                  || (In_Mode ch Operate instr && (instr.sensor_trip @ ch))
137 |
138 | Is_Ch_Tripped : Mode -> Bit -> Bit
139 | Is_Ch_Tripped mode sensor_tripped =
```

```
  // @bug abakst Constants should be replaced, but this causes
  // problems for crymp: 2 = Manual, 1 = Operate
  (mode == 2) || ((mode == 1) && sensor_tripped)

Step_Trip_Signals:
  InstrumentationUnit ->
  InstrumentationUnit
Step_Trip_Signals state =
  { state | sensor_trip = sensor_trips }
  where
    sensor_trips = Generate_Sensor_Trips state.reading state.setpoints

Saturation : [32] -> [32] -> [32]
Saturation t p = p - sat_pressure
  where sat_pressure = PressureTable t

PressureTable : [32] -> [32]
PressureTable temp = sat_pressure
  where
    idx = if temp <$ 35 then 0 else (temp-35)/5
    sat_pressure =
      if idx < 52 then Table @ idx else Table @ (51 : [lg2 52])

    // Table in 10^-5 lb/in^2
    // https://mfathi.iut.ac.ir/sites/mfathi.iut.ac.ir/files/files_course/
        ↪ table_of_saturation_vapor_0.pdf
    Table: [52][32]
    Table = [0009998,
             0012163,
             0014753,
             0017796,
             0021404,
             0025611,
             0030562,
             0036292,
             0042985,
             0050683,
             0059610,
             0069813,
             0081567,
             0094924,
             0110218,
             0127500,
             0147160,
             0169270,
             0194350,
             0222300,
             0253820,
             0288920,
             0328250,
             0371840,
             0420470,
             0474140,
             0533740,
             0599260,
             0671730,
             0751100,
             0838550,
             0934000,
             1038600,
             1152600,
             1277600,
             1413200,
             1469600,
             1718600,
             1892100,
```

```
            2079100,
            2280400,
            2496800,
            2731900,
            2984000,
            3253900,
            3542700,
            3854600,
            4187500,
            4542300,
            4920000,
            5325900,
            5775200
           ]

Generate_Sensor_Trips : [NChannels][32] -> [NChannels][32] -> [NChannels]
Generate_Sensor_Trips vals setpoints =
    // @bug abakst `2` should be `S`, but this causes problems for crymp
    // See `Is_Ch_Tripped` above as well.
    [ Trip vals setpoints 0, Trip vals setpoints 1, Trip vals setpoints 2 ]

Trip: [NChannels][32] -> [NChannels][32] -> Channel -> Bit
Trip vals setpoints ch = if ch == 2 then v <$ sp else sp < v
  where v = vals @ ch
        sp = setpoints @ ch

private
  /** @requirements
        INSTRUMENTATION_RESET
  */
  property instrumentation_reset =
    In_Maintenance Initial
    ∧ In_Mode P Bypass Initial
    ∧ In_Mode T Bypass Initial
    ∧ In_Mode S Bypass Initial

  /** @requirements
        INSTRUMENTATION_TRIP_PRESSURE
  */
  property instrumentation_trip_pressure (inp: Input) (instr: InstrumentationUnit) =
       In_Mode P Manual instr
    ∨ (In_Mode P Operate instr ∧ inp @ P > Get_Setpoint P instr')
    /* ------------------------------------------------------------------ */ ==
    (Is_Tripped P instr')
      where instr' = Handle_Input inp instr >>>> Step_Trip_Signals

  /** @requirements
        INSTRUMENTATION_TRIP_TEMPERATURE
  */
  property instrumentation_trip_temperature (inp: Input) (instr: InstrumentationUnit) =
         In_Mode T Manual instr
     ∨ (In_Mode T Operate instr ∧ inp @ T > Get_Setpoint T instr')
    /* ------------------------------------------------------------------ */ ==
    (Is_Tripped T instr')
      where instr' = Handle_Input inp instr >>>> Step_Trip_Signals

  /** @requirements
        INSTRUMENTATION_TRIP_SATURATION
  */
  property instrumentation_trip_saturation (inp: Input) (instr: InstrumentationUnit) =
       In_Mode S Manual instr
       ∨ (In_Mode S Operate instr ∧ Saturation (inp @ T) (inp @ P) <$ Get_Setpoint S instr')
    /* ------------------------------------------------------------------ */ ==
    (Is_Tripped S instr')
      where instr' = Handle_Input inp instr >>>> Step_Trip_Signals
```

```
/** @requirements
      INSTRUMENTATION_SET_MANUAL_TRIP_TEMPERATURE
      INSTRUMENTATION_SET_MANUAL_TRIP_PRESSURE
      INSTRUMENTATION_SET_MANUAL_TRIP_SATURATION
*/
property instrumentation_set_manual_trip (instr: InstrumentationUnit) =
  In_Maintenance instr ==> (
       (Is_Tripped T trippedT ∨ ~ (instr.sensor_trip @ T))
     ∧ (Is_Tripped P trippedP ∨ ~ (instr.sensor_trip @ P))
     ∧ (Is_Tripped S trippedS ∨ ~ (instr.sensor_trip @ S))
  )
  where
    trippedT = Set_Mode T Manual instr
    trippedP = Set_Mode P Manual instr
    trippedS = Set_Mode S Manual instr

/** @requirements
       INSTRUMENTATION_SET_SETPOINT_TEMPERATURE
       INSTRUMENTATION_SET_SETPOINT_PRESSURE
       INSTRUMENTATION_SET_SETPOINT_SATURATION
*/
property get_set_setpoint_correct (instr: InstrumentationUnit) (val: [32]) =
     Get_Setpoint T (Set_Setpoint T val instr) == val
  ∧ Get_Setpoint P (Set_Setpoint P val instr) == val
  ∧ Get_Setpoint S (Set_Setpoint S val instr) == val

/** @requirements
       INSTRUMENTATION_SET_BYPASS_TEMPERATURE
       INSTRUMENTATION_SET_BYPASS_PRESSURE
       INSTRUMENTATION_SET_BYPASS_SATURATION
*/
property set_bypass_correct (instr: InstrumentationUnit) =
     In_Maintenance instr ==> (
         ~ Is_Tripped T (Set_Mode T Bypass instr)
       ∧ ~ Is_Tripped P (Set_Mode P Bypass instr)
       ∧ ~ Is_Tripped S (Set_Mode S Bypass instr)
     )

property step_state_const (inp: Input) (instr: InstrumentationUnit) =
      instr.mode == instr'.mode
    ∧ instr.setpoints == instr'.setpoints
    ∧ instr.maintenance == instr'.maintenance
    where instr' = Handle_Input inp instr >>>> Step_Trip_Signals

// Not connected to a high level requirement because this simply establishes
// the connection between `Is_Tripped` and `Is_Ch_Tripped` (which is more
// convenient for synthesis)
property is_ch_trip_correct instr =
     Is_Tripped T instr == Is_Ch_Tripped (instr.mode @ T) (instr.sensor_trip @ T)
  ∧ Is_Tripped P instr == Is_Ch_Tripped (instr.mode @ P) (instr.sensor_trip @ P)
  ∧ Is_Tripped S instr == Is_Ch_Tripped (instr.mode @ S) (instr.sensor_trip @ S)
```

E.5 Cryptol Utility Functions

Listing E.5: Listing Cryptol Model of Utils.

```
// HARDENS Reactor Trip System (RTS) Utility Functions
// In support of a formal model of RTS system behavior written in the
// Cryptol DSL.
//
// @author Alex Bakst <abakst@galois.com>
// @created November, 2021
```

```
7
8  module RTS::Utils where
9
10 infixl 5 >>>>
11 (>>>>) x f = f x
12
13 numBits: {n} (fin n) => [n] -> Integer
14 numBits lst = sum [ if b then 1 else 0 | b <- lst ]
```

E.6 SAW Model

E.6.1 SAWscript to test the Actuator

Listing E.6: Listing Saw Model of actuator.

```
1  include "common.saw";
2
3  cryptol_add_path "../models";
4
5  actuator_cryp <- cryptol_load "../models/RTS/Actuator.cry";
6  actuator_gen <- llvm_load_module "generated/actuator_impl.bc";
7
8  let actuate_actuator_ref = {{ actuator_cryp::ActuateActuator }};
9
10 let actuate_actuator_spec = do {
11   input <- llvm_fresh_var "inp" (llvm_int 8);
12   let expected = {{ (zero # [actuate_actuator_ref (take (reverse input))]) : [8] }};
13   llvm_execute_func [llvm_term input];
14   llvm_return (llvm_term expected);
15 };
16
17 llvm_verify actuator_gen "ActuateActuator" [] false actuate_actuator_spec z3;
```

E.6.2 SAWscript to test the Actuation Unit

Listing E.7: Listing Saw Model of actuation_unit.

```
1  include "common.saw";
2
3  cryptol_add_path "../models";
4
5  actuate_unit_cryp <- cryptol_load "../models/RTS/ActuationUnit.cry";
6  actuate_unit_gen <- llvm_load_module "generated/actuation_unit_impl.bc";
7
8  let actuate_d0_ref = {{ actuate_unit_cryp::Actuate_D0 }};
9  let actuate_d1_ref = {{ actuate_unit_cryp::Actuate_D1 }};
10
11 let actuate_d_spec f = do {
12     (trips, ptr_trips) <- ptr_to_fresh "trips" (llvm_array 3 (llvm_array 4 (llvm_int 8)));
13     old_trip <- llvm_fresh_var "old" (llvm_int 8);
14     llvm_precond {{ (old_trip == 0) || (old_trip == 1) }};
15     llvm_execute_func [ptr_trips, llvm_term old_trip];
16     let expected = {{ (zero # [f trips (0 != old_trip)]) : [8] }};
17     llvm_return (llvm_term expected);
18 };
19
20 let actuate_d0 = actuate_d_spec actuate_d0_ref;
21 let actuate_d1 = actuate_d_spec actuate_d1_ref;
```

```
22
23 llvm_verify actuate_unit_gen "Actuate_D0" [] false actuate_d0 z3;
24 llvm_verify actuate_unit_gen "Actuate_D1" [] false actuate_d1 z3;
```

E.6.3 SAWscript to test the Instrumentation Unit

Listing E.8: Listing Saw Model of instrumentation.

```
1  include "common.saw";
2  enable_experimental;
3
4  cryptol_add_path "../models";
5
6  instrumentation_cryp <- cryptol_load "../models/RTS/InstrumentationUnit.cry";
7  instrumentation_gen <- llvm_load_module "generated/instrumentation_impl.bc";
8  instrumentation_hand <- llvm_load_module "handwritten/instrumentation_impl.bc";
9
10 let is_ch_tripped_ref = {{ instrumentation_cryp::Is_Ch_Tripped }};
11 write_verilog "Is_Ch_Tripped.v" is_ch_tripped_ref;
12
13 let generate_sensor_trips_ref = {{ instrumentation_cryp::Generate_Sensor_Trips }};
14 write_verilog "Generate_Sensor_Trips.v" generate_sensor_trips_ref;
15
16 let is_ch_tripped_spec = do {
17     mode <- llvm_fresh_var "mode" (llvm_int 8);
18     sensor_tripped <- llvm_fresh_var "sensor_tripped" (llvm_int 8);
19     llvm_precond {{ elem mode [0, 1, 2] }};
20     llvm_precond {{ elem sensor_tripped [0, 1] }};
21     llvm_execute_func [llvm_term mode, llvm_term sensor_tripped];
22     let expected = {{ (zero # [is_ch_tripped_ref (drop mode) (0 != sensor_tripped)]) : [8]
           ↪ }};
23     llvm_return (llvm_term expected);
24 };
25
26 let generate_sensor_trips_spec = do {
27     (vals, ptr_vals) <- ptr_to_fresh "vals" (llvm_array 3 (llvm_int 32));
28     (setpoints, ptr_setpoints) <- ptr_to_fresh "setpoints" (llvm_array 3 (llvm_int 32));
29     llvm_precond {{ (vals @ instrumentation_cryp::S) < 0x80000000 }};
30     llvm_execute_func [ptr_vals, ptr_setpoints];
31     let reference_val = {{ generate_sensor_trips_ref vals setpoints }};
32     let expected = {{ (zero # reverse reference_val) : [8] }};
33     llvm_return (llvm_term expected);
34 };
35
36 llvm_verify instrumentation_gen "Is_Ch_Tripped" [] false is_ch_tripped_spec z3;
37 llvm_verify instrumentation_hand "Is_Ch_Tripped" [] false is_ch_tripped_spec z3;
38 llvm_verify instrumentation_gen "Generate_Sensor_Trips" [] false generate_sensor_trips_spec
       ↪ z3;
39 llvm_verify instrumentation_hand "Generate_Sensor_Trips" [] false generate_sensor_trips_spec
       ↪  z3;
```

E.6.4 SAWscript to test the Saturation

Listing E.9: Listing Saw Model of saturation.

```
1  include "common.saw";
2  enable_experimental;
3
4  cryptol_add_path "../models";
5
```

```
instrumentation_cryp <- cryptol_load "../models/RTS/InstrumentationUnit.cry";
saturation_gen <- llvm_load_module "generated/saturation_impl.bc";

let saturation_ref = {{ instrumentation_cryp::Saturation }};

let saturation_spec = do {
    t <- llvm_fresh_var "t" (llvm_int 32);
    p <- llvm_fresh_var "p" (llvm_int 32);
    llvm_execute_func [llvm_term t, llvm_term p];
    llvm_return (llvm_term {{ saturation_ref t p }});
};

llvm_verify saturation_gen "Saturation" [] false saturation_spec z3;
```

E.6.5 SAWscript some standard definitions

Listing E.10: Listing Saw Model of common.

```
let alloc_init ty v = do {
    p <- llvm_alloc ty;
    llvm_points_to p v;
    return p;
};

let ptr_to_fresh n ty = do {
    x <- llvm_fresh_var n ty;
    p <- alloc_init ty (llvm_term x);
    return (x, p);
};
```

E.7 HDL Implementation

E.7.1 HDL model of Instrumentation

Listing E.11: Listing Verilog Model of instrumentation_impl.

```
module Is_Ch_Tripped
    #(localparam Log2Modes = 2)
    ( input logic [Log2Modes - 1:0] mode,
      input logic sensor_tripped,
      output logic out
    );
   assign out = (mode == 2) || ((mode == 1) & sensor_tripped);
endmodule

module Generate_Sensor_Trips
    #(localparam NChannels = 3)
    ( input logic [NChannels * 32 - 1:0] vals,
      input logic [NChannels * 32 - 1:0] setpoints,
      output logic [NChannels - 1:0] out
    );
   genvar ch;
   for (ch = 0; ch < NChannels; ch = ch + 1) begin
      localparam rev_ch = NChannels - ch - 1;
      logic [31:0]v = vals[(rev_ch*32) + 31 -: 32];
      logic [31:0]sp = setpoints[(rev_ch*32) + 31 -: 32];
      // SAW caught a bug here, originally used
      // `rev_ch` in the conditional
```

```
      if (ch == 2) begin
         assign out[rev_ch] = $signed(v) < $signed(sp);
      end else begin
         assign out[rev_ch] = sp < v;
      end
   end
endmodule
```

E.7.2 HDL model of Actuator

Listing E.12: Listing Verilog Model of actuator_impl.

```
module ActuateActuator
    ( input logic [1:0] inputs,
      output logic out
    );
    // ../models/RTS/Actuator.cry:31:1--31:16
    assign out = inputs[1] | inputs[0];
endmodule
```

E.7.3 HDL model of Actuation Unit

Listing E.13: Listing Verilog Model of actuation_unit_impl.

```
module Coincidence_2_4
    ( input logic [31:0] x,
      output logic out
    );
    logic a;
    logic b;
    logic c;
    logic d;
    // ../models/RTS/ActuationUnit.cry:60:7--60:8
    assign a = x[31:24] != 8'h0;
    // ../models/RTS/ActuationUnit.cry:61:7--61:8
    assign b = x[23:16] != 8'h0;
    // ../models/RTS/ActuationUnit.cry:62:7--62:8
    assign c = x[15:8] != 8'h0;
    // ../models/RTS/ActuationUnit.cry:63:7--63:8
    assign d = x[7:0] != 8'h0;
    // ../models/RTS/ActuationUnit.cry:57:3--57:18
    assign out = a & b | ((a | b) & (c | d) | c & d);
endmodule
module TemperatureLogic
    ( input logic [31:0] ts,
      output logic out
    );
    // ../models/RTS/ActuationUnit.cry:44:1--44:17
    Coincidence_2_4 Coincidence_2_4_inst1 (.x(ts),
                                           .out(out));
endmodule
module PressureLogic
    ( input logic [31:0] ts,
      output logic out
    );
    // ../models/RTS/ActuationUnit.cry:47:1--47:14
    Coincidence_2_4 Coincidence_2_4_inst1 (.x(ts),
                                           .out(out));
endmodule
module TempPressureTripOut
```

```
    ( input logic [1:0] ts,
      output logic out
    );
    // ../models/RTS/ActuationUnit.cry:53:1--53:20
    assign out = ts[1] | ts[0];
endmodule
module Actuate_D0
    ( input logic [95:0] trips,
      input logic old,
      output logic out
    );
    logic [31:0] temperatureTrips;
    logic [31:0] pressureTrips;
    logic d0;
    // ../models/RTS/ActuationUnit.cry:34:5--34:21
    assign temperatureTrips[31:0] = trips[95:64];
    // ../models/RTS/ActuationUnit.cry:35:5--35:18
    assign pressureTrips[31:0] = trips[63:32];
    // ../models/RTS/ActuationUnit.cry:32:5--32:7
    logic TemperatureLogic_out;
    TemperatureLogic TemperatureLogic_inst1 (.ts(temperatureTrips),
                                             .out(TemperatureLogic_out));
    logic PressureLogic_out;
    PressureLogic PressureLogic_inst1 (.ts(pressureTrips),
                                       .out(PressureLogic_out));
    TempPressureTripOut TempPressureTripOut_inst1 (.ts({TemperatureLogic_out,
        ↪ PressureLogic_out}),
                                                   .out(d0));
    // ../models/RTS/ActuationUnit.cry:30:1--30:11
    assign out = d0 | old;
endmodule
module SaturationLogic
    ( input logic [31:0] ts,
      output logic out
    );
    // ../models/RTS/ActuationUnit.cry:50:1--50:16
    Coincidence_2_4 Coincidence_2_4_inst1 (.x(ts),
                                           .out(out));
endmodule
module Actuate_D1
    ( input logic [95:0] trips,
      input logic old,
      output logic out
    );
    logic [31:0] saturationTrips;
    logic d1;
    // ../models/RTS/ActuationUnit.cry:41:5--41:20
    assign saturationTrips[31:0] = trips[31:0];
    // ../models/RTS/ActuationUnit.cry:40:5--40:7
    SaturationLogic SaturationLogic_inst1 (.ts(saturationTrips),
                                           .out(d1));
    // ../models/RTS/ActuationUnit.cry:38:1--38:11
    assign out = d1 | old;
endmodule
```

Listing E.14: Listing Verilog Model of nerv.

```
/*
 * NERV -- Naive Educational RISC-V Processor
 *
 * Copyright (C) 2020 Claire Xenia Wolf <claire@yosyshq.com>
 *
 * Permission to use, copy, modify, and/or distribute this software for any
 * purpose with or without fee is hereby granted, provided that the above
 * copyright notice and this permission notice appear in all copies.
 *
```

```
 * THE SOFTWARE IS PROVIDED "AS IS" AND THE AUTHOR DISCLAIMS ALL WARRANTIES
 * WITH REGARD TO THIS SOFTWARE INCLUDING ALL IMPLIED WARRANTIES OF
 * MERCHANTABILITY AND FITNESS. IN NO EVENT SHALL THE AUTHOR BE LIABLE FOR
 * ANY SPECIAL, DIRECT, INDIRECT, OR CONSEQUENTIAL DAMAGES OR ANY DAMAGES
 * WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER IN AN
 * ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING OUT OF
 * OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE.
 *
 */

module nerv #(
        parameter [31:0] RESET_ADDR = 32'h 0000_0000,
        parameter integer NUMREGS = 32
) (
        input clock,
        input reset,
        input stall,
        output trap,

`ifdef NERV_RVFI
        output reg rvfi_valid,
        output reg [63:0] rvfi_order,
        output reg [31:0] rvfi_insn,
        output reg rvfi_trap,
        output reg rvfi_halt,
        output reg rvfi_intr,
        output reg [ 1:0] rvfi_mode,
        output reg [ 1:0] rvfi_ixl,
        output reg [ 4:0] rvfi_rs1_addr,
        output reg [ 4:0] rvfi_rs2_addr,
        output reg [31:0] rvfi_rs1_rdata,
        output reg [31:0] rvfi_rs2_rdata,
        output reg [ 4:0] rvfi_rd_addr,
        output reg [31:0] rvfi_rd_wdata,
        output reg [31:0] rvfi_pc_rdata,
        output reg [31:0] rvfi_pc_wdata,
        output reg [31:0] rvfi_mem_addr,
        output reg [ 3:0] rvfi_mem_rmask,
        output reg [ 3:0] rvfi_mem_wmask,
        output reg [31:0] rvfi_mem_rdata,
        output reg [31:0] rvfi_mem_wdata,
`endif

        // we have 2 external memories
        // one is instruction memory
        output [31:0] imem_addr,
        input [31:0] imem_data,

        // the other is data memory
        output dmem_valid,
        output [31:0] dmem_addr,
        output [ 3:0] dmem_wstrb,
        output [31:0] dmem_wdata,
        input [31:0] dmem_rdata
);
        reg mem_wr_enable;
        reg [31:0] mem_wr_addr;
        reg [31:0] mem_wr_data;
        reg [3:0] mem_wr_strb;

        reg mem_rd_enable;
        reg [31:0] mem_rd_addr;
        reg [4:0] mem_rd_reg;
        reg [4:0] mem_rd_func;

        reg mem_rd_enable_q;
```

```
76        reg [4:0] mem_rd_reg_q;
77        reg [4:0] mem_rd_func_q;
78
79        // delayed copies of mem_rd
80        always @(posedge clock) begin
81               if (!stall) begin
82                      mem_rd_enable_q <= mem_rd_enable;
83                      mem_rd_reg_q <= mem_rd_reg;
84                      mem_rd_func_q <= mem_rd_func;
85               end
86               if (reset) begin
87                      mem_rd_enable_q <= 0;
88               end
89        end
90
91        // memory signals
92        assign dmem_valid = mem_wr_enable || mem_rd_enable;
93        assign dmem_addr = mem_wr_enable ? mem_wr_addr : mem_rd_enable ? mem_rd_addr : 32'h x
              ↪ ;
94        assign dmem_wstrb = mem_wr_enable ? mem_wr_strb : mem_rd_enable ? 4'h 0 : 4'h x;
95        assign dmem_wdata = mem_wr_enable ? mem_wr_data : 32'h x;
96
97        // registers, instruction reg, program counter, next pc
98        reg [31:0] regfile [0:NUMREGS-1];
99        wire [31:0] insn;
100       reg [31:0] npc;
101       reg [31:0] pc;
102
103       reg [31:0] imem_addr_q;
104
105       always @(posedge clock) begin
106              imem_addr_q <= imem_addr;
107       end
108
109       // instruction memory pointer
110       assign imem_addr = (stall || trap || mem_rd_enable_q) ? imem_addr_q : npc;
111       assign insn = imem_data;
112
113       // rs1 and rs2 are source for the instruction
114       wire [31:0] rs1_value = !insn_rs1 ? 0 : regfile[insn_rs1];
115       wire [31:0] rs2_value = !insn_rs2 ? 0 : regfile[insn_rs2];
116
117       // components of the instruction
118       wire [6:0] insn_funct7;
119       wire [4:0] insn_rs2;
120       wire [4:0] insn_rs1;
121       wire [2:0] insn_funct3;
122       wire [4:0] insn_rd;
123       wire [6:0] insn_opcode;
124
125       // split R-type instruction - see section 2.2 of RiscV spec
126       assign {insn_funct7, insn_rs2, insn_rs1, insn_funct3, insn_rd, insn_opcode} = insn;
127
128       // setup for I, S, B & J type instructions
129       // I - short immediates and loads
130       wire [11:0] imm_i;
131       assign imm_i = insn[31:20];
132
133       // S - stores
134       wire [11:0] imm_s;
135       assign imm_s[11:5] = insn_funct7, imm_s[4:0] = insn_rd;
136
137       // B - conditionals
138       wire [12:0] imm_b;
139       assign {imm_b[12], imm_b[10:5]} = insn_funct7, {imm_b[4:1], imm_b[11]} = insn_rd,
              ↪ imm_b[0] = 1'b0;
```

```
140 |
141 |       // J - unconditional jumps
142 |       wire [20:0] imm_j;
143 |       assign {imm_j[20], imm_j[10:1], imm_j[11], imm_j[19:12], imm_j[0]} = {insn[31:12], 1'
          ↪ b0};
144 |
145 |       wire [31:0] imm_i_sext = $signed(imm_i);
146 |       wire [31:0] imm_s_sext = $signed(imm_s);
147 |       wire [31:0] imm_b_sext = $signed(imm_b);
148 |       wire [31:0] imm_j_sext = $signed(imm_j);
149 |
150 |       // opcodes - see section 19 of RiscV spec
151 |       localparam OPCODE_LOAD = 7'b 00_000_11;
152 |       localparam OPCODE_STORE = 7'b 01_000_11;
153 |       localparam OPCODE_MADD = 7'b 10_000_11;
154 |       localparam OPCODE_BRANCH = 7'b 11_000_11;
155 |
156 |       localparam OPCODE_LOAD_FP = 7'b 00_001_11;
157 |       localparam OPCODE_STORE_FP = 7'b 01_001_11;
158 |       localparam OPCODE_MSUB = 7'b 10_001_11;
159 |       localparam OPCODE_JALR = 7'b 11_001_11;
160 |
161 |       localparam OPCODE_CUSTOM_0 = 7'b 00_010_11;
162 |       localparam OPCODE_CUSTOM_1 = 7'b 01_010_11;
163 |       localparam OPCODE_NMSUB = 7'b 10_010_11;
164 |       localparam OPCODE_RESERVED_0 = 7'b 11_010_11;
165 |
166 |       localparam OPCODE_MISC_MEM = 7'b 00_011_11;
167 |       localparam OPCODE_AMO = 7'b 01_011_11;
168 |       localparam OPCODE_NMADD = 7'b 10_011_11;
169 |       localparam OPCODE_JAL = 7'b 11_011_11;
170 |
171 |       localparam OPCODE_OP_IMM = 7'b 00_100_11;
172 |       localparam OPCODE_OP = 7'b 01_100_11;
173 |       localparam OPCODE_OP_FP = 7'b 10_100_11;
174 |       localparam OPCODE_SYSTEM = 7'b 11_100_11;
175 |
176 |       localparam OPCODE_AUIPC = 7'b 00_101_11;
177 |       localparam OPCODE_LUI = 7'b 01_101_11;
178 |       localparam OPCODE_RESERVED_1 = 7'b 10_101_11;
179 |       localparam OPCODE_RESERVED_2 = 7'b 11_101_11;
180 |
181 |       localparam OPCODE_OP_IMM_32 = 7'b 00_110_11;
182 |       localparam OPCODE_OP_32 = 7'b 01_110_11;
183 |       localparam OPCODE_CUSTOM_2 = 7'b 10_110_11;
184 |       localparam OPCODE_CUSTOM_3 = 7'b 11_110_11;
185 |
186 |       // next write, next destination (rd), illegal instruction registers
187 |       reg next_wr;
188 |       reg [31:0] next_rd;
189 |       reg illinsn;
190 |
191 |       reg trapped;
192 |       reg trapped_q;
193 |       assign trap = trapped;
194 |
195 |       always @* begin
196 |              // advance pc
197 |              npc = pc + 4;
198 |
199 |              // defaults for read, write
200 |              next_wr = 0;
201 |              next_rd = 0;
202 |              illinsn = 0;
203 |
204 |              mem_wr_enable = 0;
```

```
205                 mem_wr_addr = 'hx;
206                 mem_wr_data = 'hx;
207                 mem_wr_strb = 'hx;
208
209                 mem_rd_enable = 0;
210                 mem_rd_addr = 'hx;
211                 mem_rd_reg = 'hx;
212                 mem_rd_func = 'hx;
213
214                 // act on opcodes
215                 case (insn_opcode)
216                         // Load Upper Immediate
217                         OPCODE_LUI: begin
218                                 next_wr = 1;
219                                 next_rd = insn[31:12] << 12;
220                         end
221                         // Add Upper Immediate to Program Counter
222                         OPCODE_AUIPC: begin
223                                 next_wr = 1;
224                                 next_rd = (insn[31:12] << 12) + pc;
225                         end
226                         // Jump And Link (unconditional jump)
227                         OPCODE_JAL: begin
228                                 next_wr = 1;
229                                 next_rd = npc;
230                                 npc = pc + imm_j_sext;
231                                 if (npc & 32'b 11) begin
232                                         illinsn = 1;
233                                         npc = npc & ~32'b 11;
234                                 end
235                         end
236                         // Jump And Link Register (indirect jump)
237                         OPCODE_JALR: begin
238                                 case (insn_funct3)
239                                         3'b 000 /* JALR */: begin
240                                                 next_wr = 1;
241                                                 next_rd = npc;
242                                                 npc = (rs1_value + imm_i_sext) & ~32'b 1;
243                                         end
244                                         default: illinsn = 1;
245                                 endcase
246                                 if (npc & 32'b 11) begin
247                                         illinsn = 1;
248                                         npc = npc & ~32'b 11;
249                                 end
250                         end
251                         // branch instructions: Branch If Equal, Branch Not Equal, Branch Less
                                ↪ Than, Branch Greater Than, Branch Less Than Unsigned, Branch
                                ↪ Greater Than Unsigned
252                         OPCODE_BRANCH: begin
253                                 case (insn_funct3)
254                                         3'b 000 /* BEQ */: begin if (rs1_value == rs2_value) npc
                                                ↪  = pc + imm_b_sext; end
255                                         3'b 001 /* BNE */: begin if (rs1_value != rs2_value) npc
                                                ↪  = pc + imm_b_sext; end
256                                         3'b 100 /* BLT */: begin if ($signed(rs1_value) <
                                                ↪ $signed(rs2_value)) npc = pc + imm_b_sext; end
257                                         3'b 101 /* BGE */: begin if ($signed(rs1_value) >=
                                                ↪ $signed(rs2_value)) npc = pc + imm_b_sext; end
258                                         3'b 110 /* BLTU */: begin if (rs1_value < rs2_value) npc
                                                ↪  = pc + imm_b_sext; end
259                                         3'b 111 /* BGEU */: begin if (rs1_value >= rs2_value)
                                                ↪ npc = pc + imm_b_sext; end
260                                         default: illinsn = 1;
261                                 endcase
262                                 if (npc & 32'b 11) begin
263                                         illinsn = 1;
```

```
                npc = npc & ~32'b 11;
        end
end
// load from memory into rd: Load Byte, Load Halfword, Load Word, Load Byte Unsigned, Load Halfword Unsigned
OPCODE_LOAD: begin
        mem_rd_addr = rs1_value + imm_i_sext;
        casez ({insn_funct3, mem_rd_addr[1:0]})
                5'b 000_zz /* LB */,
                5'b 001_z0 /* LH */,
                5'b 010_00 /* LW */,
                5'b 100_zz /* LBU */,
                5'b 101_z0 /* LHU */: begin
                        mem_rd_enable = 1;
                        mem_rd_reg = insn_rd;
                        mem_rd_func = {mem_rd_addr[1:0], insn_funct3};
                        mem_rd_addr = {mem_rd_addr[31:2], 2'b 00};
                end
                default: illinsn = 1;
        endcase
end
// store to memory instructions: Store Byte, Store Halfword, Store Word
OPCODE_STORE: begin
        mem_wr_addr = rs1_value + imm_s_sext;
        casez ({insn_funct3, mem_wr_addr[1:0]})
                5'b 000_zz /* SB */,
                5'b 001_z0 /* SH */,
                5'b 010_00 /* SW */: begin
                        mem_wr_enable = 1;
                        mem_wr_data = rs2_value;
                        mem_wr_strb = 4'b 1111;
                        case (insn_funct3)
                                3'b 000 /* SB */: begin mem_wr_strb = 4'b 0001; end
                                3'b 001 /* SH */: begin mem_wr_strb = 4'b 0011; end
                                3'b 010 /* SW */: begin mem_wr_strb = 4'b 1111; end
                        endcase
                        mem_wr_data = mem_wr_data << (8*mem_wr_addr[1:0]) ;
                        mem_wr_strb = mem_wr_strb << mem_wr_addr[1:0];
                        mem_wr_addr = {mem_wr_addr[31:2], 2'b 00};
                end
                default: illinsn = 1;
        endcase
end
// immediate ALU instructions: Add Immediate, Set Less Than Immediate, Set Less Than Immediate Unsigned, XOR Immediate,
// OR Immediate, And Immediate, Shift Left Logical Immediate, Shift Right Logical Immediate, Shift Right Arithmetic Immediate
OPCODE_OP_IMM: begin
        casez ({insn_funct7, insn_funct3})
                10'b zzzzzzz_000 /* ADDI */: begin next_wr = 1; next_rd = rs1_value + imm_i_sext; end
                10'b zzzzzzz_010 /* SLTI */: begin next_wr = 1; next_rd = $signed(rs1_value) < $signed(imm_i_sext); end
                10'b zzzzzzz_011 /* SLTIU */: begin next_wr = 1; next_rd =  = rs1_value < imm_i_sext; end
                10'b zzzzzzz_100 /* XORI */: begin next_wr = 1; next_rd = rs1_value ^ imm_i_sext; end
                10'b zzzzzzz_110 /* ORI */: begin next_wr = 1; next_rd =  rs1_value | imm_i_sext; end
                10'b zzzzzzz_111 /* ANDI */: begin next_wr = 1; next_rd = rs1_value & imm_i_sext; end
                10'b 0000000_001 /* SLLI */: begin next_wr = 1; next_rd = rs1_value << insn[24:20]; end
```

```
317                                     10'b 0000000_101 /* SRLI */: begin next_wr = 1; next_rd
                                            ↪ = rs1_value >> insn[24:20]; end
318                                     10'b 0100000_101 /* SRAI */: begin next_wr = 1; next_rd
                                            ↪ = $signed(rs1_value) >>> insn[24:20]; end
319                                     default: illinsn = 1;
320                             endcase
321                     end
322                     OPCODE_OP: begin
323                     // ALU instructions: Add, Subtract, Shift Left Logical, Set Left Than,
                           ↪ Set Less Than Unsigned, XOR, Shift Right Logical,
324                     // Shift Right Arithmetic, OR, AND
325                             case ({insn_funct7, insn_funct3})
326                                     10'b 0000000_000 /* ADD */: begin next_wr = 1; next_rd =
                                            ↪  rs1_value + rs2_value; end
327                                     10'b 0100000_000 /* SUB */: begin next_wr = 1; next_rd =
                                            ↪  rs1_value - rs2_value; end
328                                     10'b 0000000_001 /* SLL */: begin next_wr = 1; next_rd =
                                            ↪  rs1_value << rs2_value[4:0]; end
329                                     10'b 0000000_010 /* SLT */: begin next_wr = 1; next_rd =
                                            ↪  $signed(rs1_value) < $signed(rs2_value); end
330                                     10'b 0000000_011 /* SLTU */: begin next_wr = 1; next_rd
                                            ↪ = rs1_value < rs2_value; end
331                                     10'b 0000000_100 /* XOR */: begin next_wr = 1; next_rd =
                                            ↪  rs1_value ^ rs2_value; end
332                                     10'b 0000000_101 /* SRL */: begin next_wr = 1; next_rd =
                                            ↪  rs1_value >> rs2_value[4:0]; end
333                                     10'b 0100000_101 /* SRA */: begin next_wr = 1; next_rd =
                                            ↪  $signed(rs1_value) >>> rs2_value[4:0]; end
334                                     10'b 0000000_110 /* OR */: begin next_wr = 1; next_rd =
                                            ↪ rs1_value | rs2_value; end
335                                     10'b 0000000_111 /* AND */: begin next_wr = 1; next_rd =
                                            ↪  rs1_value & rs2_value; end
336                                     default: illinsn = 1;
337                             endcase
338                     end
339                     default: illinsn = 1;
340             endcase
341
342             // if last cycle was a memory read, then this cycle is the 2nd part of it and
                   ↪ imem_data will not be a valid instruction
343             if (mem_rd_enable_q) begin
344                     npc = pc;
345                     next_wr = 0;
346                     illinsn = 0;
347                     mem_rd_enable = 0;
348                     mem_wr_enable = 0;
349             end
350
351             // reset
352             if (reset || reset_q) begin
353                     npc = RESET_ADDR;
354                     next_wr = 0;
355                     illinsn = 0;
356                     mem_rd_enable = 0;
357                     mem_wr_enable = 0;
358             end
359     end
360
361     reg reset_q;
362     reg [31:0] mem_rdata;
363 `ifdef NERV_RVFI
364     reg rvfi_pre_valid;
365     reg [ 4:0] rvfi_pre_rd_addr;
366     reg [31:0] rvfi_pre_rd_wdata;
367 `endif
368
369     // mem read functions: Lower and Upper Bytes, signed and unsigned
```

```
370         always @* begin
371                 mem_rdata = dmem_rdata >> (8*mem_rd_func_q[4:3]);
372                 case (mem_rd_func_q[2:0])
373                         3'b 000 /* LB */: begin mem_rdata = $signed(mem_rdata[7:0]); end
374                         3'b 001 /* LH */: begin mem_rdata = $signed(mem_rdata[15:0]); end
375                         3'b 100 /* LBU */: begin mem_rdata = mem_rdata[7:0]; end
376                         3'b 101 /* LHU */: begin mem_rdata = mem_rdata[15:0]; end
377                 endcase
378         end
379
380         // every cycle
381         always @(posedge clock) begin
382                 reset_q <= reset;
383                 trapped_q <= trapped;
384
385                 // increment pc if possible
386                 if (!trapped && !stall && !reset && !reset_q) begin
387                         if (illinsn)
388                                 trapped <= 1;
389                         pc <= npc;
390 `ifdef NERV_RVFI
391                         rvfi_pre_valid <= !mem_rd_enable_q;
392                         rvfi_order <= rvfi_order + 1;
393                         rvfi_insn <= insn;
394                         rvfi_trap <= illinsn;
395                         rvfi_halt <= illinsn;
396                         rvfi_intr <= 0;
397                         rvfi_mode <= 3;
398                         rvfi_ixl <= 1;
399                         rvfi_rs1_addr <= insn_rs1;
400                         rvfi_rs2_addr <= insn_rs2;
401                         rvfi_rs1_rdata <= rs1_value;
402                         rvfi_rs2_rdata <= rs2_value;
403                         rvfi_pre_rd_addr <= next_wr ? insn_rd : 0;
404                         rvfi_pre_rd_wdata <= next_wr && insn_rd ? next_rd : 0;
405                         rvfi_pc_rdata <= pc;
406                         rvfi_pc_wdata <= npc;
407                         if (dmem_valid) begin
408                                 rvfi_mem_addr <= dmem_addr;
409                                 case ({mem_rd_enable, insn_funct3})
410                                         4'b 1_000 /* LB */: begin rvfi_mem_rmask <= 4'b 0001 <<
                                                ↪ mem_rd_func[4:3]; end
411                                         4'b 1_001 /* LH */: begin rvfi_mem_rmask <= 4'b 0011 <<
                                                ↪ mem_rd_func[4:3]; end
412                                         4'b 1_010 /* LW */: begin rvfi_mem_rmask <= 4'b 1111 <<
                                                ↪ mem_rd_func[4:3]; end
413                                         4'b 1_100 /* LBU */: begin rvfi_mem_rmask <= 4'b 0001 <<
                                                ↪  mem_rd_func[4:3]; end
414                                         4'b 1_101 /* LHU */: begin rvfi_mem_rmask <= 4'b 0011 <<
                                                ↪  mem_rd_func[4:3]; end
415                                         default: rvfi_mem_rmask <= 0;
416                                 endcase
417                                 rvfi_mem_wmask <= dmem_wstrb;
418                                 rvfi_mem_wdata <= dmem_wdata;
419                         end else begin
420                                 rvfi_mem_addr <= 0;
421                                 rvfi_mem_rmask <= 0;
422                                 rvfi_mem_wmask <= 0;
423                                 rvfi_mem_wdata <= 0;
424                         end
425 `endif
426                         // update registers from memory or rd (destination)
427                         if (mem_rd_enable_q || next_wr)
428                                 regfile[mem_rd_enable_q ? mem_rd_reg_q : insn_rd] <=
                                        ↪ mem_rd_enable_q ? mem_rdata : next_rd;
429                 end
```

```
                // reset
                if (reset || reset_q) begin
                        pc <= RESET_ADDR - (reset ? 4 : 0);
                        trapped <= 0;
`ifdef NERV_RVFI
                        rvfi_pre_valid <= 0;
                        rvfi_order <= 0;
`endif
                end
        end

`ifdef NERV_RVFI
        always @* begin
                if (mem_rd_enable_q) begin
                        rvfi_rd_addr = mem_rd_reg_q;
                        rvfi_rd_wdata = mem_rd_reg_q ? mem_rdata : 0;
                end else begin
                        rvfi_rd_addr = rvfi_pre_rd_addr;
                        rvfi_rd_wdata = rvfi_pre_rd_wdata;
                end
                rvfi_valid = rvfi_pre_valid && !stall && !reset && !reset_q && !trapped_q;
                rvfi_mem_rdata = dmem_rdata;
        end
`endif

`ifdef NERV_DBGREGS
        wire [31:0] dbg_reg_x0 = 0;
        wire [31:0] dbg_reg_x1 = regfile[1];
        wire [31:0] dbg_reg_x2 = regfile[2];
        wire [31:0] dbg_reg_x3 = regfile[3];
        wire [31:0] dbg_reg_x4 = regfile[4];
        wire [31:0] dbg_reg_x5 = regfile[5];
        wire [31:0] dbg_reg_x6 = regfile[6];
        wire [31:0] dbg_reg_x7 = regfile[7];
        wire [31:0] dbg_reg_x8 = regfile[8];
        wire [31:0] dbg_reg_x9 = regfile[9];
        wire [31:0] dbg_reg_x10 = regfile[10];
        wire [31:0] dbg_reg_x11 = regfile[11];
        wire [31:0] dbg_reg_x12 = regfile[12];
        wire [31:0] dbg_reg_x13 = regfile[13];
        wire [31:0] dbg_reg_x14 = regfile[14];
        wire [31:0] dbg_reg_x15 = regfile[15];
        wire [31:0] dbg_reg_x16 = regfile[16];
        wire [31:0] dbg_reg_x17 = regfile[17];
        wire [31:0] dbg_reg_x18 = regfile[18];
        wire [31:0] dbg_reg_x19 = regfile[19];
        wire [31:0] dbg_reg_x20 = regfile[20];
        wire [31:0] dbg_reg_x21 = regfile[21];
        wire [31:0] dbg_reg_x22 = regfile[22];
        wire [31:0] dbg_reg_x23 = regfile[23];
        wire [31:0] dbg_reg_x24 = regfile[24];
        wire [31:0] dbg_reg_x25 = regfile[25];
        wire [31:0] dbg_reg_x26 = regfile[26];
        wire [31:0] dbg_reg_x27 = regfile[27];
        wire [31:0] dbg_reg_x28 = regfile[28];
        wire [31:0] dbg_reg_x29 = regfile[29];
        wire [31:0] dbg_reg_x30 = regfile[30];
        wire [31:0] dbg_reg_x31 = regfile[31];
`endif
endmodule
```

Listing E.15: Listing Verilog Model of testbench.

```
/*
 * NERV -- Naive Educational RISC-V Processor
```

```
 *
 * Copyright (C) 2020 N. Engelhardt <nak@yosyshq.com>
 *
 * Permission to use, copy, modify, and/or distribute this software for any
 * purpose with or without fee is hereby granted, provided that the above
 * copyright notice and this permission notice appear in all copies.
 *
 * THE SOFTWARE IS PROVIDED "AS IS" AND THE AUTHOR DISCLAIMS ALL WARRANTIES
 * WITH REGARD TO THIS SOFTWARE INCLUDING ALL IMPLIED WARRANTIES OF
 * MERCHANTABILITY AND FITNESS. IN NO EVENT SHALL THE AUTHOR BE LIABLE FOR
 * ANY SPECIAL, DIRECT, INDIRECT, OR CONSEQUENTIAL DAMAGES OR ANY DAMAGES
 * WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER IN AN
 * ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING OUT OF
 * OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE.
 *
 */

module testbench;

localparam TIMEOUT = (1<<10);
reg clock;

wire LEDR_N, LEDG_N, LED1, LED2, LED3, LED4, LED5;

always #5 clock = clock === 1'b0;

top dut (
        .CLK(clock),
        .LEDR_N(LEDR_N),
        .LEDG_N(LEDG_N),
        .LED1(LED1),
        .LED2(LED2),
        .LED3(LED3),
        .LED4(LED4),
        .LED5(LED5)
);

initial begin
        if ($test$plusargs("vcd")) begin
                $dumpfile("testbench.vcd");
                $dumpvars(0, testbench);
        end
end

reg [31:0] cycles = 0;

always @(posedge clock) begin
        cycles <= cycles + 32'h1;
        if (cycles >= TIMEOUT) begin
                $display("Simulated %0d cycles", cycles);
                $finish;
        end
end

endmodule
```

Listing E.16: Listing Verilog Model of testbench.

```
/*
 * NERV -- Naive Educational RISC-V Processor
 *
 * Copyright (C) 2020 N. Engelhardt <nak@yosyshq.com>
 *
 * Permission to use, copy, modify, and/or distribute this software for any
 * purpose with or without fee is hereby granted, provided that the above
 * copyright notice and this permission notice appear in all copies.
```

```
 *
 * THE SOFTWARE IS PROVIDED "AS IS" AND THE AUTHOR DISCLAIMS ALL WARRANTIES
 * WITH REGARD TO THIS SOFTWARE INCLUDING ALL IMPLIED WARRANTIES OF
 * MERCHANTABILITY AND FITNESS. IN NO EVENT SHALL THE AUTHOR BE LIABLE FOR
 * ANY SPECIAL, DIRECT, INDIRECT, OR CONSEQUENTIAL DAMAGES OR ANY DAMAGES
 * WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER IN AN
 * ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING OUT OF
 * OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE.
 *
 */

module testbench;

localparam TIMEOUT = (1<<10);
reg clock;

wire LEDR_N, LEDG_N, LED1, LED2, LED3, LED4, LED5;

always #5 clock = clock === 1'b0;

top dut (
        .CLK(clock),
        .LEDR_N(LEDR_N),
        .LEDG_N(LEDG_N),
        .LED1(LED1),
        .LED2(LED2),
        .LED3(LED3),
        .LED4(LED4),
        .LED5(LED5)
);

initial begin
        if ($test$plusargs("vcd")) begin
                $dumpfile("testbench.vcd");
                $dumpvars(0, testbench);
        end
end

reg [31:0] cycles = 0;

always @(posedge clock) begin
        cycles <= cycles + 32'h1;
        if (cycles >= TIMEOUT) begin
                $display("Simulated %0d cycles", cycles);
                $finish;
        end
end

endmodule
```

Listing E.17: Listing Verilog Model of wrapper.

```
/*
 * NERV -- Naive Educational RISC-V Processor
 *
 * Copyright (C) 2020 Claire Xenia Wolf <claire@yosyshq.com>
 *
 * Permission to use, copy, modify, and/or distribute this software for any
 * purpose with or without fee is hereby granted, provided that the above
 * copyright notice and this permission notice appear in all copies.
 *
 * THE SOFTWARE IS PROVIDED "AS IS" AND THE AUTHOR DISCLAIMS ALL WARRANTIES
 * WITH REGARD TO THIS SOFTWARE INCLUDING ALL IMPLIED WARRANTIES OF
 * MERCHANTABILITY AND FITNESS. IN NO EVENT SHALL THE AUTHOR BE LIABLE FOR
 * ANY SPECIAL, DIRECT, INDIRECT, OR CONSEQUENTIAL DAMAGES OR ANY DAMAGES
 * WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER IN AN
```

```
 * ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING OUT OF
 * OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE.
 *
 */

module rvfi_wrapper (
	input clock,
	input reset,
	`RVFI_OUTPUTS
);
	(* keep *) `rvformal_rand_reg stall;
	(* keep *) `rvformal_rand_reg [31:0] imem_data;
	(* keep *) `rvformal_rand_reg [31:0] dmem_rdata;

	(* keep *) wire [31:0] imem_addr;

	(* keep *) wire dmem_valid;
	(* keep *) wire [31:0] dmem_addr;
	(* keep *) wire [ 3:0] dmem_wstrb;
	(* keep *) wire [31:0] dmem_wdata;

	nerv uut (
		.clock (clock ),
		.reset (reset ),
		.stall (stall ),

		.imem_addr (imem_addr ),
		.imem_data (imem_data ),

		.dmem_valid (dmem_valid),
		.dmem_addr (dmem_addr ),
		.dmem_wstrb (dmem_wstrb),
		.dmem_wdata (dmem_wdata),
		.dmem_rdata (dmem_rdata),

		`RVFI_CONN
	);

`ifdef NERV_FAIRNESS
	reg [2:0] stalled = 0;
	always @(posedge clock) begin
		stalled <= {stalled, stall};
		assume (~stalled);
	end
`endif
endmodule
```

Listing E.18: Listing Verilog Model of instrumentation_impl.

```
module Is_Ch_Tripped
    #(localparam Log2Modes = 2)
    ( input logic [Log2Modes - 1:0] mode,
      input logic sensor_tripped,
      output logic out
    );
   assign out = (mode == 2) || ((mode == 1) & sensor_tripped);
endmodule

module Generate_Sensor_Trips
    #(localparam NChannels = 3)
    ( input logic [NChannels * 32 - 1:0] vals,
      input logic [NChannels * 32 - 1:0] setpoints,
      output logic [NChannels - 1:0] out
    );
   genvar ch;
   for (ch = 0; ch < NChannels; ch = ch + 1) begin
```

```
      localparam rev_ch = NChannels - ch - 1;
      logic [31:0]v = vals[(rev_ch*32) + 31 -: 32];
      logic [31:0]sp = setpoints[(rev_ch*32) + 31 -: 32];
      // SAW caught a bug here, originally used
      // `rev_ch` in the conditional
      if (ch == 2) begin
         assign out[rev_ch] = $signed(v) < $signed(sp);
      end else begin
         assign out[rev_ch] = sp < v;
      end
   end
endmodule
```

Listing E.19: Listing Verilog Model of testbench.

```
/*
 * NERV -- Naive Educational RISC-V Processor
 *
 * Copyright (C) 2020 N. Engelhardt <nak@yosyshq.com>
 *
 * Permission to use, copy, modify, and/or distribute this software for any
 * purpose with or without fee is hereby granted, provided that the above
 * copyright notice and this permission notice appear in all copies.
 *
 * THE SOFTWARE IS PROVIDED "AS IS" AND THE AUTHOR DISCLAIMS ALL WARRANTIES
 * WITH REGARD TO THIS SOFTWARE INCLUDING ALL IMPLIED WARRANTIES OF
 * MERCHANTABILITY AND FITNESS. IN NO EVENT SHALL THE AUTHOR BE LIABLE FOR
 * ANY SPECIAL, DIRECT, INDIRECT, OR CONSEQUENTIAL DAMAGES OR ANY DAMAGES
 * WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER IN AN
 * ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING OUT OF
 * OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE.
 *
 */

module testbench;

localparam TIMEOUT = (1<<10);
reg clock;

wire LEDR_N, LEDG_N, LED1, LED2, LED3, LED4, LED5;

always #5 clock = clock === 1'b0;

top dut (
	.CLK(clock),
	.LEDR_N(LEDR_N),
	.LEDG_N(LEDG_N),
	.LED1(LED1),
	.LED2(LED2),
	.LED3(LED3),
	.LED4(LED4),
	.LED5(LED5)
);

initial begin
	if ($test$plusargs("vcd")) begin
		$dumpfile("testbench.vcd");
		$dumpvars(0, testbench);
	end
end

reg [31:0] cycles = 0;

always @(posedge clock) begin
	cycles <= cycles + 32'h1;
	if (cycles >= TIMEOUT) begin
```

```
52              $display("Simulated %0d cycles", cycles);
53              $finish;
54      end
55  end
56
57  endmodule
```

Listing E.20: Listing Verilog Model of nervsoc.

```
1   /*
2    * NERV -- Naive Educational RISC-V Processor
3    *
4    * Copyright (C) 2020 Claire Xenia Wolf <claire@yosyshq.com>
5    *
6    * Permission to use, copy, modify, and/or distribute this software for any
7    * purpose with or without fee is hereby granted, provided that the above
8    * copyright notice and this permission notice appear in all copies.
9    *
10   * THE SOFTWARE IS PROVIDED "AS IS" AND THE AUTHOR DISCLAIMS ALL WARRANTIES
11   * WITH REGARD TO THIS SOFTWARE INCLUDING ALL IMPLIED WARRANTIES OF
12   * MERCHANTABILITY AND FITNESS. IN NO EVENT SHALL THE AUTHOR BE LIABLE FOR
13   * ANY SPECIAL, DIRECT, INDIRECT, OR CONSEQUENTIAL DAMAGES OR ANY DAMAGES
14   * WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER IN AN
15   * ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING OUT OF
16   * OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE.
17   *
18   */
19
20  module nervsoc (
21          input clock,
22          input reset,
23          output reg [31:0] leds
24  );
25          reg [31:0] imem [0:1023];
26          reg [31:0] dmem [0:1023];
27
28          wire stall = 0;
29          wire trap;
30
31          wire [31:0] imem_addr;
32          reg [31:0] imem_data;
33
34          wire dmem_valid;
35          wire [31:0] dmem_addr;
36          wire [3:0] dmem_wstrb;
37          wire [31:0] dmem_wdata;
38          reg [31:0] dmem_rdata;
39
40          initial begin
41                  $readmemh("firmware.hex", imem);
42          end
43
44          always @(posedge clock)
45                  imem_data <= imem[imem_addr[31:2]];
46
47          always @(posedge clock) begin
48                  if (dmem_valid) begin
49                          if (dmem_addr == 32'h 0100_0000) begin
50                                  if (dmem_wstrb[0]) leds[ 7: 0] <= dmem_wdata[ 7: 0];
51                                  if (dmem_wstrb[1]) leds[15: 8] <= dmem_wdata[15: 8];
52                                  if (dmem_wstrb[2]) leds[23:16] <= dmem_wdata[23:16];
53                                  if (dmem_wstrb[3]) leds[31:24] <= dmem_wdata[31:24];
54                          end else begin
55                                  if (dmem_wstrb[0]) dmem[dmem_addr[31:2]][ 7: 0] <= dmem_wdata[
                                        ↪ 7: 0];
```

```
                            if (dmem_wstrb[1]) dmem[dmem_addr[31:2]][15: 8] <= dmem_wdata
                                ↪ [15: 8];
                            if (dmem_wstrb[2]) dmem[dmem_addr[31:2]][23:16] <= dmem_wdata
                                ↪ [23:16];
                            if (dmem_wstrb[3]) dmem[dmem_addr[31:2]][31:24] <= dmem_wdata
                                ↪ [31:24];
                        end
                        dmem_rdata <= dmem[dmem_addr[31:2]];
                end
        end

        nerv cpu (
                .clock (clock ),
                .reset (reset ),
                .stall (stall ),
                .trap (trap ),

                .imem_addr (imem_addr ),
                .imem_data (imem_data ),

                .dmem_valid(dmem_valid),
                .dmem_addr (dmem_addr ),
                .dmem_wstrb(dmem_wstrb),
                .dmem_wdata(dmem_wdata),
                .dmem_rdata(dmem_rdata)
        );
endmodule
```

E.8 BlueSpec Implementations

Listing E.21: Listing Verilog Model of FpgaTop.

```
import NervSoC :: *;

// IO
import I2C :: *;
import RS232 :: *;
import GetPut::*;

interface FpgaTop_IFC;
    (* prefix = "" *)
    interface RS232 rs232;
    //(* prefix = "" *)
    //interface I2C_Pins i2c;
    (* always_ready *)
    method Bit#(8) leds();
endinterface

(* synthesize *)
module mkFpgaTop_IFC(FpgaTop_IFC);
    // Divisor of 10 for baudrate of 76800
    UART #(4) uart <- mkUART(8, NONE, STOP_1, 10);
    //I2CController #(1) i2c_controller <- mkI2CController();

    NervSoC_IFC nerv_soc <- mkNervSoC;

    rule uart_transmit;
        let val <- nerv_soc.get_uart_tx_byte();
        uart.rx.put(val);
    endrule

    Reg #(Bit #(12)) rg_console_in_poll <- mkReg (0);
```

```
    rule uart_receive;
        if (rg_console_in_poll == 0) begin
           Bit #(8) ch <- uart.tx.get();
           if (ch != 0) begin
              nerv_soc.set_uart_rx_byte(ch);
           end
        end
        rg_console_in_poll <= rg_console_in_poll + 1;
    endrule



    interface rs232 = uart.rs232;
    // TODO: i2c interface
    //interface I2C_Pins i2c = i2c_controller.i2c;
    method leds = nerv_soc.gpio;
endmodule
```

Listing E.22: Listing Verilog Model of Actuation_Generated_BVI.

```
package Actuation_Generated_BVI;

import Clocks :: *;
import Actuation::*;

(* synthesize *)
module mkActuationGenerated(Actuation_IFC);
    ActuationD0_IFC a0 <- mkActuationGeneratedD0();
    ActuationD1_IFC a1 <- mkActuationGeneratedD1();

    interface ActuationD0_IFC d0;
        method actuate_d0 = a0.actuate_d0();
    endinterface
    interface ActuationD1_IFC d1;
        method actuate_d1 = a1.actuate_d1();
    endinterface
endmodule

import "BVI" Actuate_D0 =
module mkActuationGeneratedD0 (ActuationD0_IFC);
    default_clock ();
    default_reset ();

    method out actuate_d0 (trips, old);
    schedule (actuate_d0) CF (actuate_d0);
endmodule

import "BVI" Actuate_D1 =
module mkActuationGeneratedD1 (ActuationD1_IFC);
    default_clock ();
    default_reset ();

    method out actuate_d1 (trips, old);
    schedule (actuate_d1) CF (actuate_d1);
endmodule

endpackage
```

Listing E.23: Listing Verilog Model of Nerv_BVI.

```
// Copyright (c) 2022 Rishiyur S. Nikhil

package Nerv_BVI;

```

```
// ================================================================
// Module mkNerv_BVI imports the SystemVerilog module nerv.sv so that
// it can be used in BSV.
// nerv.sv is from: https://github.com/YosysHQ/nerv

// ================================================================
// Import from BSV library

import Clocks :: *;

// ================================================================
// BSV "raw" interface for nerv.sv (nerv CPU)

interface Nerv_BVI_IFC;
   (* always_ready, always_enabled *)
   method Action m_stall (Bool b);
   (* always_ready *)
   method Bool m_trap;

   (* always_ready *)
   method Bit #(32) m_imem_addr;
   (* always_ready, always_enabled *)
   method Action m_imem_data (Bit #(32) xi);

   (* always_ready *)
   method Bool m_dmem_valid;
   (* always_ready *)
   method Bit #(32) m_dmem_addr;
   (* always_ready *)
   method Bit #(4) m_dmem_wstrb;
   (* always_ready *)
   method Bit #(32) m_dmem_wdata;
   (* always_ready, always_enabled *)
   method Action m_dmem_rdata (Bit #(32) xd);
endinterface

// ================================================================

import "BVI" nerv =
module mkNerv_BVI (Nerv_BVI_IFC);
   default_clock (clock);

   // BSV's default reset (including this module's reset) is active low
   // The imported nerv.sv's reset is active high.
   default_reset (reset) <- invertCurrentReset;

   method m_stall (stall) enable ((*inhigh*) EN0); // BSV -> Verilog
   method trap m_trap; // BSV <- Verilog

   method imem_addr m_imem_addr; // BSV <- Verilog
   method m_imem_data (imem_data) enable ((*inhigh*) EN1); // BSV -> Verilog

   method dmem_valid m_dmem_valid; // BSV <- Verilog
   method dmem_addr m_dmem_addr; // BSV <- Verilog
   method dmem_wstrb m_dmem_wstrb; // BSV <- Verilog
   method dmem_wdata m_dmem_wdata; // BSV <- Verilog
   method m_dmem_rdata (dmem_rdata) enable ((*inhigh*) EN2); // BSV -> Verilog

   // Declaring each method conflict-free with each method.
   // This is ok if everything is registered, but dicey if there are
   // combinational paths weaving in and out of the module.
      schedule
      (m_stall,
       m_trap,
       m_imem_addr,
       m_imem_data,
```

```
      m_dmem_valid,
      m_dmem_addr,
      m_dmem_wstrb,
      m_dmem_wdata,
      m_dmem_rdata) CF (m_stall,
                        m_trap,
                        m_imem_addr,
                        m_imem_data,
                        m_dmem_valid,
                        m_dmem_addr,
                        m_dmem_wstrb,
                        m_dmem_wdata,
                        m_dmem_rdata);
endmodule

// ================================================================

endpackage
```

Listing E.24: Listing Verilog Model of Nerv.

```
// Copyright (c) 2022 Rishiyur S. Nikhil

package Nerv;

// ================================================================
// Module mkNerv is a thin wrapper around mkNerv_BVI to make DMem
// outputs (strobe and data) into a struct

// ================================================================
// Import from BSV library

// None

// ================================================================
// Local imports

import Nerv_BVI :: *;

// ================================================================
// Interface for nerv as we might like to see it in BSV

typedef struct {
   Bit #(4) wstrb;
   Bit #(32) wdata;
   }
DmemWrite
deriving (Bits, Eq, FShow);

interface Nerv_IFC;
   (* always_ready, always_enabled *)
   method Action m_stall (Bool b);
   (* always_ready *)
   method Bool m_trap;

   (* always_ready *)
   method Bit #(32) m_imem_addr;
   (* always_ready, always_enabled *)
   method Action m_imem_data (Bit #(32) xi);

   method Bit #(32) m_dmem_addr;
   method DmemWrite m_get_dmem;

   (* always_ready, always_enabled *)
   method Action m_dmem_rdata (Bit #(32) xd);

```

```
   method Bool m_dmem_valid;
endinterface

// ================================================================

(* synthesize *)
module mkNerv (Nerv_IFC);

   Nerv_BVI_IFC nerv_BVI <- mkNerv_BVI;

   method Action m_stall (Bool b) = nerv_BVI.m_stall (b);

   method Bool m_trap = nerv_BVI.m_trap;
   method Bool m_dmem_valid = nerv_BVI.m_dmem_valid;

   method Bit #(32) m_imem_addr = nerv_BVI.m_imem_addr;
   method Action m_imem_data (Bit #(32) xi) = nerv_BVI.m_imem_data (xi);

   method Bit #(32) m_dmem_addr () if (nerv_BVI.m_dmem_valid);
      return nerv_BVI.m_dmem_addr;
   endmethod

   method DmemWrite m_get_dmem () if (nerv_BVI.m_dmem_valid);
      return DmemWrite {wstrb: nerv_BVI.m_dmem_wstrb,
                        wdata: nerv_BVI.m_dmem_wdata};
   endmethod

   method Action m_dmem_rdata (Bit #(32) xd) = nerv_BVI.m_dmem_rdata (xd);
endmodule

// ================================================================

endpackage
```

Listing E.25: Listing Verilog Model of Instrumentation_Handwritten_BVI.

```
package Instrumentation_Handwritten_BVI;

import Clocks :: *;
import Instrumentation::*;

(* synthesize *)
module mkInstrumentationHandwritten(Instrumentation_IFC);
    ChannelTripped_IFC i_channel <- mkInstrHandwrittenIsChannelTripped();
    SensorTrips_IFC i_sensors <- mkInstrHandwrittenGenerateSensorTrips();

    interface ChannelTripped_IFC channel;
        method is_channel_tripped = i_channel.is_channel_tripped();
    endinterface
    interface SensorTrips_IFC sensors;
        method generate_sensor_trips = i_sensors.generate_sensor_trips();
    endinterface
endmodule

import "BVI" Is_Ch_Tripped_Handwritten =
module mkInstrHandwrittenIsChannelTripped (ChannelTripped_IFC);
    default_clock ();
    default_reset ();

    method out is_channel_tripped (mode, sensor_tripped);
    schedule (is_channel_tripped) CF (is_channel_tripped);
endmodule

import "BVI" Generate_Sensor_Trips_Handwritten =
module mkInstrHandwrittenGenerateSensorTrips (SensorTrips_IFC);
    default_clock ();
```

```
    default_reset ();

    method out generate_sensor_trips (vals, setpoints);
    schedule (generate_sensor_trips) CF (generate_sensor_trips);
endmodule

endpackage
```

Listing E.26: Listing Verilog Model of Instrumentation.

```
package Instrumentation;

import Vector :: *;

// Instrumentation interface
interface Instrumentation_IFC;
    interface ChannelTripped_IFC channel;
    interface SensorTrips_IFC sensors;
endinterface

// Sub-interface for each implemented function
interface ChannelTripped_IFC;
    (* always_ready *)
    method Bool is_channel_tripped (Bit#(2) mode,
                                    Bool sensor_tripped);
endinterface

interface SensorTrips_IFC;
    (* always_ready *)
    method Bit#(3) generate_sensor_trips (Vector#(3, Bit#(32)) vals,
                                          Vector#(3, Bit#(32)) setpoints);
endinterface

endpackage
```

Listing E.27: Listing Verilog Model of Top.

```
// Copyright (c) 2022 Rishiyur S. Nikhil

// ================================================================
// Top-level module instantiates NervSoC and displays the LED outputs
// whenever they change.

// ================================================================
// Import from BSV library

// None

// ================================================================
// Local imports

import NervSoC :: *;

// IO
import I2C :: *;
import RS232 :: *;
import GetPut::*;

// ================================================================
// Top

import "BDPI" function Action c_putchar (Bit #(8) c);
import "BDPI" function ActionValue #(Bit #(8)) c_trygetchar (Bit #(8) dummy);
import "BDPI" function ActionValue #(Bit #(8)) c_i2c_request (Bit #(8) addr,
```

```
                                                    Bit #(8) data);

(* clock_prefix="CLK", reset_prefix="RST_N" *)
(* synthesize *)
module mkTop (Empty);

   Reg #(Bit #(8)) rg_gpio <- mkReg (0);

   NervSoC_IFC nerv_soc <- mkNervSoC;

   // I/O peripherals
   // @podhrmic TODO: check the prescalers
   // and look into proper use of I2C module
   I2CController #(1) i2c_controller <- mkI2CController();
   UART #(4) uart <- mkUART(8, NONE, STOP_1, 16);
   //uart.RS232.sout ?

   // ================================================================
   // UART console I/O
   // Based on https://github.com/bluespec/Piccolo/blob/master/src_Testbench/Top/Top_HW_Side.bsv
   //
   // Poll terminal input and relay any chars into system console input.
   // Note: rg_console_in_poll is used to poll only every N cycles, whenever it wraps around to 0.
   // Note: if the SoC starts dropping bytes, try increasing the register size
   Reg #(Bit #(12)) rg_console_in_poll <- mkReg (0);
`ifdef SIMULATION
   begin
   rule uart_rx;
      if (rg_console_in_poll == 0) begin
         Bit #(8) ch <- c_trygetchar (?);
         if (ch != 0) begin
            nerv_soc.set_uart_rx_byte(ch);
         end
      end
      rg_console_in_poll <= rg_console_in_poll + 1;
   endrule
   end
`else
   // FPGA
   begin
   rule uart_rx;
      if (rg_console_in_poll == 0) begin
         Bit #(8) ch <- uart.tx.get();
         if (ch != 0) begin
            nerv_soc.set_uart_rx_byte(ch);
         end
      end
      rg_console_in_poll <= rg_console_in_poll + 1;
   endrule
   end
`endif

   rule uart_tx;
      let val <- nerv_soc.get_uart_tx_byte();
`ifdef SIMULATION
      c_putchar(val);
`else
      uart.rx.put(val);
`endif
   endrule

   Reg #(Bit #(8)) rg_i2c_resp <- mkRegU();
   Reg #(Bool) rg_i2c_complete <- mkReg(False);
   rule i2c_request(!rg_i2c_complete);
```

```
92      let request <- nerv_soc.i2c_get_request();
93      let val <- c_i2c_request(request.address, request.data);
94      rg_i2c_resp <= val;
95      rg_i2c_complete <= True;
96   endrule
97
98   rule i2c_response(rg_i2c_complete);
99      let response = I2CResponse { data: rg_i2c_resp};
100     nerv_soc.i2c_give_response(response);
101     rg_i2c_complete <= False;
102  endrule
103
104  rule rl_leds;
105     let gpio = nerv_soc.gpio;
106     if (gpio != rg_gpio) $display ("GPIO: %032b", gpio);
107     rg_gpio <= gpio;
108  endrule
109
110endmodule
111
112// ================================================================
```

Listing E.28: Listing Verilog Model of NervSoC.

```
1  // Copyright (c) 2022 Rishiyur S. Nikhil, Michal Podhradsky
2  package NervSoC;
3
4  // ================================================================
5  // Module mkNervSoC is a BSV version of nervsoc.sv
6
7  // nervsoc.sv is from: https://github.com/YosysHQ/nerv
8
9  // ================================================================
10 // Import from BSV library
11 import RegFile :: *;
12 import Vector :: *;
13 import I2C :: *;
14 import BRAM :: *;
15 import BRAMCore :: *;
16
17 // ================================================================
18 // Local imports
19 import Nerv :: *;
20 import Instrumentation::*;
21 import Instrumentation_Handwritten_BVI::*;
22 import Instrumentation_Generated_BVI::*;
23 import Actuation::*;
24 import Actuation_Generated_BVI::*;
25 // ================================================================
26 // A small NERV SoC
27
28 interface NervSoC_IFC;
29    // This sets the name of the result
30    method Bit #(8) gpio;
31    // TX -> a byte to be send
32    method ActionValue#(Bit #(8)) get_uart_tx_byte;
33    // RX -> a byte to be received
34    method Action set_uart_rx_byte(Bit #(8) rx);
35    // I2C methods
36    method ActionValue #(I2CRequest) i2c_get_request;
37    method Action i2c_give_response(I2CResponse r);
38 endinterface
39
40 // ================================================================
41 typedef enum { REQ_I, PUSH_I, REQ_D, PUSH_D, STOP } State deriving(Bits,Eq);
42 (* synthesize *)
```

```
43  module mkNervSoC (NervSoC_IFC);
44      /**
45      * ////////////////////////////////////////////////////////////////
46      * Instantiate interfaces
47      * ////////////////////////////////////////////////////////////////
48      */
49      Nerv_IFC nerv <- mkNerv;
50      Instrumentation_IFC instr_hand <- mkInstrumentationHandwritten();
51      Instrumentation_IFC instr_gen <- mkInstrumentationGenerated();
52      Actuation_IFC actuation_gen <- mkActuationGenerated();
53
54      // For debugging only
55      Bool show_exec_trace = False;
56      Bool show_load_store = False;
57
58      /**
59      * ////////////////////////////////////////////////////////////////
60      * IO memory map
61      * ////////////////////////////////////////////////////////////////
62      */
63      Bit #(32) gpio_addr = 32'h 0100_0000;
64
65      Bit #(32) uart_reg_addr_tx = 32'h 0200_0000;
66      Bit #(32) uart_reg_addr_rx = 32'h 0200_0004;
67      Bit #(32) uart_reg_addr_dr = 32'h 0200_0008;
68
69      Bit #(32) i2c_reg_addr_base = 32'h 0300_0000;
70      Bit #(32) i2c_reg_addr_data = 32'h 0300_0004; // I2C fifo has up to 16 bytes (4 registers)
71      Bit #(32) i2c_reg_addr_stat = 32'h 0300_0008; // I2C status reg (transaction complete 1bit
            ↪ , transaction error 1bit, error type 2bits)
72
73      Bit #(32) clock_reg_adrr_lower = 32'h 0400_0000; // System ticks
74      Bit #(32) clock_reg_adrr_upper = 32'h 0400_0004;
75
76      Bit #(32) instr_reg_addr_hand_base = 32'h 0500_0000; // Handwritten Instrumentation base
            ↪ register
77      Bit #(32) instr_reg_addr_hand_instr_val_0 = 32'h 0500_0004;
78      Bit #(32) instr_reg_addr_hand_instr_val_1 = 32'h 0500_0008;
79      Bit #(32) instr_reg_addr_hand_instr_val_2 = 32'h 0500_000C;
80      Bit #(32) instr_reg_addr_hand_setpoint_val_0 = 32'h 0500_0010;
81      Bit #(32) instr_reg_addr_hand_setpoint_val_1 = 32'h 0500_0014;
82      Bit #(32) instr_reg_addr_hand_setpoint_val_2 = 32'h 0500_0018;
83      Bit #(32) instr_reg_addr_hand_res = 32'h 0500_001C;
84
85      Bit #(32) instr_reg_addr_gen_base = 32'h 0500_0020; // Generated Instrumentation base
            ↪ register
86      Bit #(32) instr_reg_addr_gen_instr_val_0 = 32'h 0500_0024;
87      Bit #(32) instr_reg_addr_gen_instr_val_1 = 32'h 0500_0028;
88      Bit #(32) instr_reg_addr_gen_instr_val_2 = 32'h 0500_002C;
89      Bit #(32) instr_reg_addr_gen_setpoint_val_0 = 32'h 0500_0030;
90      Bit #(32) instr_reg_addr_gen_setpoint_val_1 = 32'h 0500_0034;
91      Bit #(32) instr_reg_addr_gen_setpoint_val_2 = 32'h 0500_0038;
92      Bit #(32) instr_reg_addr_gen_res = 32'h 0500_003C;
93
94      Bit #(32) actuation_reg_addr_gen_base = 32'h 0500_0040; // Generated Actuation base
            ↪ register
95      Bit #(32) actuation_reg_addr_gen_trip_0 = 32'h 0500_0044;
96      Bit #(32) actuation_reg_addr_gen_trip_1 = 32'h 0500_0048;
97      Bit #(32) actuation_reg_addr_gen_trip_2 = 32'h 0500_004C;
98      Bit #(32) actuation_reg_addr_gen_res = 32'h 0500_0050;
99
100     Bit #(32) io_top_addr = 32'h 0500_0050;
101
102     /**
103     * ////////////////////////////////////////////////////////////////
104     * IO registers
```

```
105   * ////////////////////////////////////////////////////////////////
106   */
107   Reg #(Bit #(32)) rg_gpio <- mkReg(0);
108   Reg #(Bit #(8)) rg_uart_tx <- mkReg(0);
109   Reg #(Bit #(8)) rg_uart_rx <- mkReg(0);
110   Reg #(Bool) rg_uart_rx_data_ready <- mkReg(False);
111   Reg #(Bool) rg_uart_tx_data_ready <- mkReg(False);
112   Reg #(Bit #(8)) rg_i2c_addr <- mkReg(0);
113   Reg #(Bit #(32)) rg_i2c_data <- mkReg(0);
114   Reg #(Bit #(32)) rg_i2c_status <- mkReg(0);
115   Reg #(Bool) rg_i2c_transaction_ready <- mkReg(False);
116   Reg #(Bit #(32)) rg_i2c_transaction_complete <- mkReg(0);
117   Vector#(3, Reg#(Bit #(32))) instr_hand_vals <- replicateM( mkReg(0) );
118   Vector#(3, Reg#(Bit #(32))) instr_hand_setpoints <- replicateM( mkReg(0) );
119   Vector#(3, Reg#(Bit #(32))) instr_gen_vals <- replicateM( mkReg(0) );
120   Vector#(3, Reg#(Bit #(32))) instr_gen_setpoints <- replicateM( mkReg(0) );
121   Vector#(3, Reg#(Bit #(32))) actuation_trips <- replicateM( mkReg(0) );
122   Reg #(Bit #(32)) rg_actuation_res <- mkReg(0);
123   Reg #(Bit #(32)) rg_instr_hand_res <- mkReg(0);
124   Reg #(Bit #(32)) rg_instr_gen_res <- mkReg(0);
125
126   RWire#(Bit #(64)) rw_tick <- mkRWire();
127   Reg#(Bit#(30)) rg_dmem_addr <- mkReg(0);
128   Reg#(Bit#(32)) rg_dmem_put_data <- mkReg(0);
129
130   /**
131   * ////////////////////////////////////////////////////////////////
132   * Memory Definition
133   * ////////////////////////////////////////////////////////////////
134   */
135   // Memory size
136   Integer imemory_size = 'h07000;
137   Integer dmemory_size = 'h07000;
138
139   // Nerv has Harward architecture (separate data and instruction memory),
140   // so in order to properly initialize global symbols, we need to load
141   // the hex file into *both* memories.
142   // NOTE: BRAM has size defined as `reg [DATA_WIDTH-1:0] RAM[0:MEMSIZE-1];`
143   // while RegFileLoad was `reg [data_width - 1 : 0] arr[lo:hi];`
144   // The size+1 is simply to make the current hex file fit.
145   BRAM_PORT#(Bit#(30), Bit#(32)) dmem_bram <- mkBRAMCore1Load(dmemory_size+1, False,"
          ↪ dmem_contents.memhex32", False);
146   BRAM_PORT#(Bit#(30), Bit#(32)) imem_bram <- mkBRAMCore1Load(imemory_size+1, False,"
          ↪ imem_contents.memhex32", False);
147
148   Reg #(Bit #(32)) rg_imem_addr <- mkReg (0);
149   Reg #(Bit #(32)) rg_imem_data <- mkRegU;
150   Reg #(Bit #(32)) rg_dmem_rdata <- mkRegU;
151   Reg #(Bool) rg_update_dmem <- mkReg(False);
152   Reg #(Bit #(64)) rg_tick <- mkReg (0);
153   Reg#(State) state <- mkReg(REQ_I);
154
155   /**
156   * ////////////////////////////////////////////////////////////////
157   * Function definitions
158   * ////////////////////////////////////////////////////////////////
159   */
160   function Bit #(8) strb2byte (Bit #(1) b) = signExtend (b);
161
162   // GPIO update
163   function ActionValue#(Bit#(32)) fn_gpio(Bit#(32) mask, Bit#(32) wdata)
164     = actionvalue
165       let gpio_val = ((rg_gpio & (~ mask)) | (wdata & mask));
166       rg_gpio <= gpio_val;
167       return gpio_val;
168     endactionvalue;
```

```
169 |
170 |    // UART
171 |    function ActionValue#(Bit#(32)) fn_uart(Bit#(32) addr, Bit#(32) wdata)
172 |       = actionvalue
173 |          case (addr)
174 |             // Write a byte to serial port
175 |             uart_reg_addr_tx:
176 |                begin
177 |                   rg_uart_tx <= wdata[7:0];
178 |                   rg_uart_tx_data_ready <= True;
179 |                   return signExtend(wdata[7:0]);
180 |                end
181 |             // Receive data from serial port
182 |             // Note: might be 0 or stale, check uart_reg_addr_dr first
183 |             uart_reg_addr_rx:
184 |                begin
185 |                   rg_uart_rx_data_ready <= False;
186 |                   return signExtend(rg_uart_rx);
187 |                end
188 |             uart_reg_addr_dr:
189 |                begin
190 |                   if (rg_uart_rx_data_ready)
191 |                      return 1;
192 |                   else
193 |                      return 0;
194 |                end
195 |             default:
196 |                return 'hFFFF;
197 |          endcase
198 |       endactionvalue;
199 |
200 |    // I2C
201 |    function ActionValue#(Bit#(32)) fn_i2c(Bit#(32) addr, Bit#(32) mask, Bit#(32) wdata)
202 |       = actionvalue
203 |          case (addr)
204 |             i2c_reg_addr_base:
205 |                begin
206 |                   // Only 8 bytes for the address, the rest is ignored
207 |                   rg_i2c_addr <= wdata[7:0];
208 |                   rg_i2c_transaction_ready <= True;
209 |                   return wdata;
210 |                end
211 |             i2c_reg_addr_data:
212 |                begin
213 |                   if (mask == 0)
214 |                   begin
215 |                      // Read rg_i2c_data
216 |                      return rg_i2c_data;
217 |                   end
218 |                   else
219 |                   begin
220 |                      // Write to rg_i2c_data
221 |                      rg_i2c_data <= wdata;
222 |                      return wdata;
223 |                   end
224 |                end
225 |             i2c_reg_addr_stat:
226 |                begin
227 |                   rg_i2c_transaction_complete <= 0;
228 |                   return rg_i2c_transaction_complete;
229 |                end
230 |          endcase
231 |       endactionvalue;
232 |
233 |    // Clock
234 |    function Bit#(32) fn_clock(Bit#(32) addr);
```

```
235       if (addr == clock_reg_adrr_lower)
236          return rg_tick[31:0];
237       else
238          return rg_tick[63:32];
239    endfunction
240
241    // Instrumentation handwritten
242    function ActionValue#(Bit#(32)) fn_instrumentation_handwritten(Bit#(32) addr, Bit#(32)
           ↪ mask, Bit#(32) wdata)
243    = actionvalue
244       let val = 0;
245       case (addr)
246          instr_reg_addr_hand_base:
247             begin
248             // wdata[0] - fnc select ( 0 - is_channel_tripped | 1 - generate_sensor_trips)
249             // wdata[2:1] - mode
250             // wdata[3] - sensor_tripped
251             // rg_instr_hand_res[2:0] - result
252             // rg_instr_hand_res[31] - fnc select ( 0 - is_channel_tripped | 1 -
                    ↪ generate_sensor_trips)
253             if (wdata[0] == 0)
254                begin
255                   // is_channel_tripped
256                   // method Bool is_channel_tripped (Bit #(2) mode, Bool sensor_tripped);
257                   let mode = wdata[2:1];
258                   let sensor_tripped = unpack(wdata[3]);
259                   rg_instr_hand_res <= signExtend( pack(instr_hand.channel.is_channel_tripped(
                          ↪ mode, sensor_tripped)) );
260                end
261             else
262                begin
263                   // generate_sensor_trips
264                   // NOTE: the values and setpoints are in reverse order.
265                   Vector#(3, Bit#(32)) vals = newVector;
266                   vals[2] = instr_hand_vals[0];
267                   vals[1] = instr_hand_vals[1];
268                   vals[0] = instr_hand_vals[2];
269
270                   Vector#(3, Bit#(32)) setpoints = newVector;
271                   setpoints[2] = instr_hand_setpoints[0];
272                   setpoints[1] = instr_hand_setpoints[1];
273                   setpoints[0] = instr_hand_setpoints[2];
274
275                   let res = signExtend(pack(
276                              instr_hand.sensors.generate_sensor_trips(vals, setpoints)
277                           ));
278                   res[31] = 1;
279                   rg_instr_hand_res <= res;
280                end
281             end
282          instr_reg_addr_hand_instr_val_0:
283             begin
284                instr_hand_vals[0] <= ((instr_hand_vals[0] & (~ mask)) | (wdata & mask));
285             end
286          instr_reg_addr_hand_instr_val_1:
287             begin
288                instr_hand_vals[1] <= ((instr_hand_vals[1] & (~ mask)) | (wdata & mask));
289             end
290          instr_reg_addr_hand_instr_val_2:
291             begin
292                instr_hand_vals[2] <= ((instr_hand_vals[2] & (~ mask)) | (wdata & mask));
293             end
294          instr_reg_addr_hand_setpoint_val_0:
295             begin
296                instr_hand_setpoints[0] <= ((instr_hand_setpoints[0] & (~ mask)) | (wdata &
                       ↪ mask));
```

```
297             end
298          instr_reg_addr_hand_setpoint_val_1:
299             begin
300                instr_hand_setpoints[1] <= ((instr_hand_setpoints[1] & (~ mask)) | (wdata &
                       ↪ mask));
301             end
302          instr_reg_addr_hand_setpoint_val_2:
303             begin
304                instr_hand_setpoints[2] <= ((instr_hand_setpoints[2] & (~ mask)) | (wdata &
                       ↪ mask));
305             end
306          instr_reg_addr_hand_res:
307             begin
308                val = rg_instr_hand_res;
309             end
310       endcase
311       return val;
312    endactionvalue;
313
314    // Instrumentation generated
315    function ActionValue#(Bit#(32)) fn_instrumentation_generated(Bit#(32) addr, Bit#(32) mask,
           ↪  Bit#(32) wdata)
316    = actionvalue
317       let val = 0;
318       case (addr)
319          instr_reg_addr_gen_base:
320             begin
321             // wdata[0] - fnc select ( 0 - is_channel_tripped | 1 - generate_sensor_trips)
322             // wdata[2:1] - mode
323             // wdata[3] - sensor_tripped
324             // rg_instr_gen_res[2:0] - result
325             // rg_instr_gen_res[31] - fnc select ( 0 - is_channel_tripped | 1 -
                       ↪ generate_sensor_trips)
326             if (wdata[0] == 0)
327                begin
328                // is_channel_tripped
329                // method Bool is_channel_tripped (Bit #(2) mode, Bool sensor_tripped);
330                let mode = wdata[2:1];
331                let sensor_tripped = unpack(wdata[3]);
332                rg_instr_gen_res <= zeroExtend(pack(
333                                   instr_gen.channel.is_channel_tripped(mode, sensor_tripped)
334                                   ));
335                end
336             else
337                begin
338                   // generate_sensor_trips
339                   // NOTE: the values and setpoints are in reverse order.
340                   Vector#(3, Bit#(32)) vals = newVector;
341                   vals[2] = instr_gen_vals[0];
342                   vals[1] = instr_gen_vals[1];
343                   vals[0] = instr_gen_vals[2];
344
345                   Vector#(3, Bit#(32)) setpoints = newVector;
346                   setpoints[2] = instr_gen_setpoints[0];
347                   setpoints[1] = instr_gen_setpoints[1];
348                   setpoints[0] = instr_gen_setpoints[2];
349                   let res = zeroExtend(pack(
350                             instr_gen.sensors.generate_sensor_trips(vals, setpoints)
351                           ));
352                   res[31] = 1;
353                   rg_instr_gen_res <= res;
354                end
355             end
356          instr_reg_addr_gen_instr_val_0:
357             begin
358                instr_gen_vals[0] <= ((instr_gen_vals[0] & (~ mask)) | (wdata & mask));
```

```
359            end
360         instr_reg_addr_gen_instr_val_1:
361            begin
362               instr_gen_vals[1] <= ((instr_gen_vals[1] & (~ mask)) | (wdata & mask));
363            end
364         instr_reg_addr_gen_instr_val_2:
365            begin
366               instr_gen_vals[2] <= ((instr_gen_vals[2] & (~ mask)) | (wdata & mask));
367            end
368         instr_reg_addr_gen_setpoint_val_0:
369            begin
370               instr_gen_setpoints[0] <= ((instr_gen_setpoints[0] & (~ mask)) | (wdata & mask)
                     ↪ );
371            end
372         instr_reg_addr_gen_setpoint_val_1:
373            begin
374               instr_gen_setpoints[1] <= ((instr_gen_setpoints[1] & (~ mask)) | (wdata & mask)
                     ↪ );
375            end
376         instr_reg_addr_gen_setpoint_val_2:
377            begin
378               instr_gen_setpoints[2] <= ((instr_gen_setpoints[2] & (~ mask)) | (wdata & mask)
                     ↪ );
379            end
380         instr_reg_addr_gen_res:
381            begin
382               val = rg_instr_gen_res;
383            end
384      endcase
385      return val;
386   endactionvalue;
387
388   // Actuation Generated
389   function ActionValue#(Bit#(32)) fn_actuation(Bit#(32) addr, Bit#(32) mask, Bit#(32) wdata)
390      = actionvalue
391      let val = 0;
392      case (addr)
393         actuation_reg_addr_gen_base:
394            begin
395               // base - trigger the actuation
396               Bool old = unpack(wdata[0]);
397               Vector#(3, Bit#(32)) trips = newVector;
398               trips[0] = actuation_trips[0];
399               trips[1] = actuation_trips[1];
400               trips[2] = actuation_trips[2];
401               // wdata[0] - value of `old` argument
402               // wdata[1] - which actuator to actuate
403               if (wdata[1] == 0)
404                  begin
405                     // Actuate D0
406                     rg_actuation_res <= zeroExtend( pack(actuation_gen.d0.actuate_d0(trips,
                           ↪ old)) );
407                  end
408               else
409                  begin
410                     // Actuate D1
411                     rg_actuation_res <= zeroExtend( pack(actuation_gen.d1.actuate_d1(trips,
                           ↪ old)) );
412                  end
413            end
414         actuation_reg_addr_gen_trip_0:
415            begin
416               // Set value for trip value 0
417               actuation_trips[0] <= ((actuation_trips[0] & (~ mask)) | (wdata & mask));
418            end
419         actuation_reg_addr_gen_trip_1:
```

```
420            begin
421               // Set value for trip value 1
422               actuation_trips[1] <= ((actuation_trips[1] & (~ mask)) | (wdata & mask));
423            end
424         actuation_reg_addr_gen_trip_2:
425            begin
426               // Set value for trip value 2
427               actuation_trips[2] <= ((actuation_trips[2] & (~ mask)) | (wdata & mask));
428            end
429         actuation_reg_addr_gen_res:
430            begin
431               // Get actuation results
432               val = rg_actuation_res;
433            end
434      endcase
435      return val;
436      endactionvalue;
437
438   /**
439   * /////////////////////////////////////////////////////////////////
440   * State machine
441   * /////////////////////////////////////////////////////////////////
442   */
443   // default state: request a new instruction from m_imem_addr
444   rule stateReqI (state == REQ_I);
445      imem_bram.put(False, nerv.m_imem_addr [31:2], 0);
446      nerv.m_stall (True); // stall CPU until the fetch is done
447      rg_tick <= rg_tick + 1;
448      nerv.m_dmem_rdata (rg_dmem_rdata);
449      state <= PUSH_I;
450   endrule
451
452   // push the new instruction from the memory to the CPU
453   rule statePushI (state == PUSH_I);
454      nerv.m_imem_data (imem_bram.read());
455      if (nerv.m_dmem_valid)
456         state <= REQ_D;
457      else
458         state <= STOP;
459   endrule
460
461   // request data from a new data memory address
462   rule stateReqD (state == REQ_D);
463      dmem_bram.put(False, nerv.m_dmem_addr [31:2], 0);
464      state <= PUSH_D;
465   endrule
466
467   // push new data into the CPU
468   rule statePushD (state == PUSH_D);
469      let d_addr = nerv.m_dmem_addr;
470      let mem_data = dmem_bram.read();
471      let dmw = nerv.m_get_dmem;
472      let wstrb = dmw.wstrb;
473      let wdata = dmw.wdata;
474      let mask = {strb2byte (wstrb [3]),
475          strb2byte (wstrb [2]),
476          strb2byte (wstrb [1]),
477          strb2byte (wstrb [0])};
478
479      let put_data = ((mem_data & (~ mask)) | (wdata & mask));
480      if (show_load_store)
481         $display ("DMem addr 0x%0h wstrb 0x%0h wdata 0x%0h mask 0x%0h put_data 0x%0h" ,
               ↪ d_addr[31:2], wstrb, wdata, mask, put_data);
482
483      // a priority encoder that takes the first arm whose condition is true.
484      case (True)
```

```
485         // GPIO update
486         (gpio_addr == d_addr):
487            put_data <- fn_gpio(mask, wdata);
488         // UART
489         (uart_reg_addr_tx <= d_addr && d_addr < i2c_reg_addr_base):
490            put_data <- fn_uart(d_addr, wdata);
491         // I2C
492         (i2c_reg_addr_base <= d_addr && d_addr < clock_reg_adrr_lower):
493            put_data <- fn_i2c(d_addr, mask, wdata);
494         // Clock
495         (clock_reg_adrr_lower <= d_addr && d_addr < instr_reg_addr_hand_base):
496            put_data = fn_clock(d_addr);
497         // Instrumentation handwritten
498         (instr_reg_addr_hand_base <= d_addr && d_addr < instr_reg_addr_gen_base):
499            put_data <- fn_instrumentation_handwritten(d_addr, mask, wdata);
500         // Instrumentation generated
501         (instr_reg_addr_gen_base <= d_addr && d_addr < actuation_reg_addr_gen_base):
502            put_data <- fn_instrumentation_generated(d_addr, mask, wdata);
503         // Actuation Generated
504         (actuation_reg_addr_gen_base <= d_addr && d_addr <= io_top_addr):
505            put_data <- fn_actuation(d_addr, mask, wdata);
506         default:
507            // Regular memory read (no IO)
508            begin
509               dmem_bram.put(True, d_addr [31:2], put_data);
510            end
511      endcase
512      // RDATA are always updated
513      rg_dmem_rdata <= put_data;
514      state <= STOP;
515   endrule
516
517   rule stateStop (state == STOP);
518      nerv.m_dmem_rdata (rg_dmem_rdata);
519      nerv.m_stall (False); // un-stall the CPU
520      state <= REQ_I;
521   endrule
522
523   /**
524   * ////////////////////////////////////////////////////////////////
525   * Terminate if trapped
526   * ////////////////////////////////////////////////////////////////
527   */
528   rule trap;
529      if (nerv.m_trap)
530      begin
531         $display ("Trapped");
532         $finish(0);
533      end
534   endrule
535
536   /**
537   * ////////////////////////////////////////////////////////////////
538   * SOC Interface
539   * ////////////////////////////////////////////////////////////////
540   */
541   // set GPIO
542   method Bit #(8) gpio = rg_gpio[7:0];
543
544   // TX -> a byte to be send
545   method ActionValue#(Bit #(8)) get_uart_tx_byte () if (rg_uart_tx_data_ready);
546      begin
547         rg_uart_tx_data_ready <= False;
548         return rg_uart_tx;
549      end
550   endmethod
```

```
551
552     // Rx -> a byte to be received
553     method Action set_uart_rx_byte(Bit #(8) rx);
554        begin
555           rg_uart_rx_data_ready <= True;
556           rg_uart_rx <= rx;
557        end
558     endmethod
559
560     // I2C methods
561     method ActionValue #(I2CRequest) i2c_get_request () if (rg_i2c_transaction_ready);
562        begin
563           let r = I2CRequest {
564                write: unpack(rg_i2c_addr[0]),
565                address: rg_i2c_addr[7:0], // Unclear what this is for
566                slaveaddr: rg_i2c_addr[7:1],
567                data: rg_i2c_data[7:0]
568             };
569           rg_i2c_transaction_ready <= False;
570           return r;
571        end
572     endmethod
573
574     method Action i2c_give_response(I2CResponse r);
575        begin
576           rg_i2c_data <= signExtend(r.data);
577           rg_i2c_transaction_complete <= 1;
578        end
579     endmethod
580  endmodule
581
582  // ========================================================
583
584  endpackage
```

Listing E.29: Listing Verilog Model of Actuation.

```
1   package Actuation;
2
3   import Vector :: *;
4
5   // Actuation interface
6   interface Actuation_IFC;
7       interface ActuationD0_IFC d0;
8       interface ActuationD1_IFC d1;
9   endinterface
10
11  interface ActuationD0_IFC;
12      (* always_ready *)
13      method Bool actuate_d0 (Vector#(3, Bit#(32)) trips,
14                              Bool old);
15  endinterface
16
17  interface ActuationD1_IFC;
18      (* always_ready *)
19      method Bool actuate_d1 (Vector#(3, Bit#(32)) trips,
20                              Bool old);
21  endinterface
22
23  endpackage
```

Listing E.30: Listing Verilog Model of Instrumentation_Generated_BVI.

```
1   package Instrumentation_Generated_BVI;
```

```
2
3  import Clocks :: *;
4  import Instrumentation::*;
5
6  (* synthesize *)
7  module mkInstrumentationGenerated(Instrumentation_IFC);
8      ChannelTripped_IFC i_channel <- mkInstrGeneratedIsChannelTripped();
9      SensorTrips_IFC i_sensors <- mkInstrGeneratedGenerateSensorTrips();
10
11     interface ChannelTripped_IFC channel;
12         method is_channel_tripped = i_channel.is_channel_tripped();
13     endinterface
14     interface SensorTrips_IFC sensors;
15         method generate_sensor_trips = i_sensors.generate_sensor_trips();
16     endinterface
17 endmodule
18
19 import "BVI" Is_Ch_Tripped_Generated =
20 module mkInstrGeneratedIsChannelTripped (ChannelTripped_IFC);
21     default_clock ();
22     default_reset ();
23
24     method out is_channel_tripped (mode, sensor_tripped);
25     schedule (is_channel_tripped) CF (is_channel_tripped);
26 endmodule
27
28 import "BVI" Generate_Sensor_Trips_Generated =
29 module mkInstrGeneratedGenerateSensorTrips (SensorTrips_IFC);
30     default_clock ();
31     default_reset ();
32
33     method out generate_sensor_trips (vals, setpoints);
34     schedule (generate_sensor_trips) CF (generate_sensor_trips);
35 endmodule
36
37 endpackage
```

E.9 ACSL Model

E.9.1 ACSL Model of the Actuation Unit

Listing E.31: Listing C Model of models.

```
1  #ifndef MODELS_ACSL_
2  #define MODELS_ACSL_
3  #include <stdint.h>
4
5  /* axiomatic Actuator {
6
7     // Refines RTS::Actuator::ActuateActuator
8     logic boolean ActuateActuator(uint8_t input) =
9       ((input & 0x1) != 0) || ((input & 0x2) != 0);
10    }
11
12    axiomatic ActuationUnit {
13
14    // Refines RTS::ActuationUnit::Coincidence_2_4
15    logic boolean Coincidence_2_4(uint8_t *trips) =
16      \let a = trips[0] != 0;
17      \let b = trips[1] != 0;
18      \let c = trips[2] != 0;
19      \let d = trips[3] != 0;
```

```
    (a&&b) || ((a||b) && (c||d)) || (c&&d);

  // Refines RTS::ActuationUnit::Actuate_D0
  logic boolean Actuate_D0(uint8_t *tripsT, uint8_t *tripsP, uint8_t *tripsS, boolean old) =
    Coincidence_2_4(tripsT) || Coincidence_2_4(tripsP) || old;

  // Refines RTS::ActuationUnit::Actuate_D1
  logic boolean Actuate_D1(uint8_t *tripsT, uint8_t *tripsP, uint8_t *tripsS, boolean old) =
    Coincidence_2_4(tripsS) || old;

  }


  axiomatic Instrumentation {

  // Refines RTS::InstrumentationUnit::Trip
  logic boolean Trip(uint32_t *vals, uint32_t *setpoints, integer channel) =
    channel == 2 ? ((int)vals[channel] < (int)setpoints[channel])
                 : (setpoints[channel] < vals[channel]);

  // Refines RTS::InstrumentationUnit::Generate_Sensor_Trips
  logic integer Generate_Sensor_Trips(uint32_t *vals, uint32_t *setpoints) =
    \let t = Trip(vals, setpoints, T);
    \let p = Trip(vals, setpoints, P);
    \let s = Trip(vals, setpoints, S);
    (t ? 1 : 0) + (p ? 2 : 0) + (s ? 4 : 0);

  // Refines RTS::InstrumentationUnit::Is_Ch_Tripped
  logic boolean Is_Ch_Tripped(integer mode, boolean tripped) =
    (mode == 2) || ((mode == 1) && tripped);

  }
 */
#endif
```

Listing E.32: Various common definitions.

```
#ifndef COMMON_H_
#define COMMON_H_

#include <stdint.h>

////////////////////////////////////////////////////////////
// Constants derived from architecture and Cryptol model //
////////////////////////////////////////////////////////////

// Instrumentation
// Trip modes:
#define NINSTR 4
#define NMODES 3
#define BYPASS 0
#define OPERATE 1
#define TRIP 2

// Command Types
#define SET_MODE 0
#define SET_MAINTENANCE 1
#define SET_SETPOINT 2

// Channel/Trip signal IDs
#define NTRIP 3
#define T 0
#define P 1
#define S 2

// Actuation
```

```
#define NVOTE_LOGIC 2
#define NDEV 2

// Core
// Command Types
#define INSTRUMENTATION_COMMAND 0
#define ACTUATION_COMMAND 1

#define BIT(_test, _value) ((_test) ? (0x8 | (_value)) : _value)
#define VALID(_value) (!(0x8 & (_value)))
#define VAL(_value) (0x1 & value)

#define NLINES 21
#define LINELENGTH 64
////////////////////////////////////////////////////////////
// RTS Command Definitions //
////////////////////////////////////////////////////////////

// Instrumentation
struct set_mode {
  uint8_t channel;
  uint8_t mode_val;
};
struct set_maintenance {
  uint8_t on;
};
struct set_setpoint {
  uint8_t channel;
  uint32_t val;
};
struct instrumentation_command {
  uint8_t type;
  uint8_t valid;
  union {
    struct set_mode mode;
    struct set_maintenance maintenance;
    struct set_setpoint setpoint;
  } cmd;
};

// Actuation
struct actuation_command {
    uint8_t device;
    uint8_t on;
};

// Root command structure
struct rts_command {
  uint8_t type;
  uint8_t instrumentation_division;
  union {
    struct instrumentation_command instrumentation;
    struct actuation_command act;
  } cmd;
};

// Redefine variable bit-width types:
#define _ExtInt_1 char
#define _ExtInt_2 char
#define _ExtInt_3 char
#define _ExtInt_4 char
#define _ExtInt_6 char
#define _ExtInt_8 char
#define _ExtInt_32 int
#define _ExtInt(w) _ExtInt_##w

```

```
// Generate names for implementation variants
#define VARIANT(source,lang,f) VARIANT_IMPL(source,lang,f)
#define VARIANT_IMPL(source,lang,f) f ## _ ## source ## _ ## lang
#define VARIANT_IMPL2(source,lang,f) source ## lang ## f

#endif // COMMON_H_
```

Listing E.33: Various core definitions.

```
#ifndef CORE_H_
#define CORE_H_

#include "common.h"

#ifndef SELF_TEST_PERIOD_SEC
#define SELF_TEST_PERIOD_SEC 20
#endif

#define NDIVISIONS 4

#ifndef T_THRESHOLD // degrees F
#define T_THRESHOLD 3
#endif

#ifndef P_THRESHOLD // 10^-5 lb/in^2
#define P_THRESHOLD 100
#endif

struct ui_values {
  uint32_t values[NDIVISIONS][NTRIP];
  uint8_t bypass[NDIVISIONS][NTRIP];
  uint8_t trip[NDIVISIONS][NTRIP];
  uint8_t maintenance[NDIVISIONS];
  char display[NLINES][LINELENGTH+1];

  uint8_t actuators[2][NDEV];
};

struct test_state {
  uint32_t test;
  uint32_t test_timer_start;
  uint8_t self_test_running;
  uint8_t self_test_expect;
  uint8_t failed;

  uint8_t test_device_result[2];

  uint8_t test_instrumentation[2];
  uint8_t test_actuation_unit;
  uint8_t test_device;

  uint8_t test_instrumentation_done[4];
  uint8_t test_actuation_unit_done[2];
  uint8_t test_device_done[2];

  uint32_t test_setpoints[4][3];
  uint32_t test_inputs[4][2];

  uint8_t actuation_old_vote;
};

struct core_state {
  struct ui_values ui;
  struct test_state test;
  uint8_t error;
};
```

```
58
59 extern struct core_state core;
60
61 int set_display_line(struct ui_values *ui, uint8_t line_number, char *display, uint32_t size
       ↪ );
62
63 void core_init(struct core_state *core);
64 int core_step(struct core_state *core);
65 #endif // CORE_H_
```

Listing E.34: The actuate interface.

```
1  #ifndef ACTUATE_H_
2  #define ACTUATE_H_
3
4  #include <stdint.h>
5  #include "models.acsl"
6
7  // Combine the votes from both actuate logic components
8  // and tell the hardware device to actuate (or unactuate)
9  int actuate_devices(void);
10
11 // Return whether or not a device with the provided votes should be actuated
12 // Bit i = vote by logic unit i
13 // This function is generated directly from the Cryptol model
14 /*@ assigns \nothing;
15   @ ensures \result == 0 || \result == 1;
16   @ ensures \result == 1 <==> ((vs & 0x01) || (vs & 0x02));
17   @ ensures ActuateActuator(vs) <==> \result == 1;
18 */
19 uint8_t ActuateActuator(uint8_t vs);
20
21 int actuate_devices_generated_C(void);
22
23 #endif // ACTUATE_H_
```

Listing E.35: Interface of the actuation logic.

```
1  #ifndef ACTUATION_H_
2  #define ACTUATION_H_
3
4  #include "stdint.h"
5  #include "common.h"
6  #include "instrumentation.h"
7  #include "core.h"
8  #include "models.acsl"
9
10 /*@requires \valid(&trips[0.. NINSTR -1]);
11   @assigns \nothing;
12   @ensures (\result != 0) <==> Coincidence_2_4(trips);
13 */
14 uint8_t Coincidence_2_4(uint8_t trips[4]);
15
16 /*@requires \valid(&trips[0.. NTRIP - 1][0.. NINSTR - 1]);
17   @requires \valid(trips + (0.. NTRIP-1));
18   @assigns \nothing;
19   @ensures (\result != 0) <==> Actuate_D0(&trips[T][0], &trips[P][0], &trips[S][0], old != 0);
20 */
21 uint8_t Actuate_D0(uint8_t trips[3][4], uint8_t old);
22
23 /*@requires \valid(&trips[0.. NTRIP-1][0.. NINSTR-1]);
24   @requires \valid(trips + (0.. NTRIP-1));
25   @assigns \nothing;
26   @ensures (\result != 0) <==> Actuate_D1(&trips[T][0], &trips[P][0], &trips[S][0], old != 0);
27 */
```

```
uint8_t Actuate_D1(uint8_t trips[3][4], uint8_t old);

struct actuation_logic {
    uint8_t vote_actuate[NDEV];
    uint8_t manual_actuate[NDEV];
};

extern struct actuation_logic actuation_logic[2];

/* The main logic of the actuation unit */

/*@requires \valid(state);
  @requires logic_no <= 1;
  @assigns state->manual_actuate[0.. NDEV-1];
  @assigns state->vote_actuate[0.. NDEV-1];
  @assigns core.test.actuation_old_vote;
  @assigns core.test.test_actuation_unit_done[logic_no];
*/
int actuation_unit_step(uint8_t logic_no, struct actuation_logic *state);

#endif // ACTUATION_H_
```

Listing E.36: Instrumentation definitions.

```
#ifndef INSTRUMENTATION_H_
#define INSTRUMENTATION_H_

#include "common.h"
#include "core.h"
#include "models.acsl"

#define ShouldTrip(_vals, _setpoints, _ch) \
  ((_ch == T && _vals[T] > _setpoints[T]) \
   || (_ch == P && _vals[P] > _setpoints[P]) \
   || (_ch == S && (int)_vals[S] < (int)_setpoints[S]))

/*@ assigns \nothing; */
uint32_t Saturation(uint32_t x, uint32_t y);

/*@requires \valid(vals + (0.. NTRIP-1));
  @requires \valid(setpoints + (0.. NTRIP-1));
  @assigns \nothing;
  @ensures \result == (uint8_t)Generate_Sensor_Trips(vals, setpoints);
*/
uint8_t Generate_Sensor_Trips(uint32_t vals[3], uint32_t setpoints[3]);

/*@requires \valid(vals + (0.. NTRIP-1));
  @requires \valid(setpoints + (0.. NTRIP-1));
  @requires ch < NTRIP;
  @assigns \nothing;
  @ensures \result == 0 || \result == 1;
  @ensures (\result == 1) <==> Trip(vals, setpoints, ch);
*/
uint8_t Trip(uint32_t vals[3], uint32_t setpoints[3], uint8_t ch);

/*@requires mode < NMODES;
  @requires trip <= 1;
  @assigns \nothing;
  @ensures (\result != 0) <==> Is_Ch_Tripped(mode, trip != 0);
*/
uint8_t Is_Ch_Tripped(uint8_t mode, uint8_t trip);

struct instrumentation_state {
  uint32_t reading[NTRIP];
  uint32_t test_reading[NTRIP];
  uint32_t setpoints[NTRIP];
```

```
  uint8_t sensor_trip[NTRIP];
  uint8_t mode[NTRIP];
  uint8_t maintenance;
  uint8_t test_complete;
};

void instrumentation_init(struct instrumentation_state *state);

/*@requires \valid(state);
  @requires \valid(state->reading + (0.. NTRIP-1));
  @requires \valid(state->test_reading + (0.. NTRIP-1));
  @requires \valid(state->setpoints + (0.. NTRIP-1));
  @requires \valid(state->sensor_trip + (0.. NTRIP-1));
  @requires state->mode[T] \in {BYPASS, OPERATE, TRIP};
  @requires state->mode[P] \in {BYPASS, OPERATE, TRIP};
  @requires state->mode[S] \in {BYPASS, OPERATE, TRIP};
  @requires div < NTRIP;
  @assigns state->reading[0.. NTRIP-1];
  @assigns state->test_reading[0.. NTRIP-1];
  @assigns state->setpoints[0.. NTRIP-1];
  @assigns state->sensor_trip[0.. NTRIP-1];
  @assigns state->maintenance;
  @assigns state->mode[0.. NTRIP-1];
  @assigns core.test.test_instrumentation_done[div];
  @ensures state->mode[T] \in {BYPASS, OPERATE, TRIP};
  @ensures state->mode[P] \in {BYPASS, OPERATE, TRIP};
  @ensures state->mode[S] \in {BYPASS, OPERATE, TRIP};
*/
int instrumentation_step(uint8_t div, struct instrumentation_state *state);

#endif // INSTRUMENTATION_H_
```

Listing E.37: Listing Interface Model of platform.

```
#ifndef PLATFORM_H_
#define PLATFORM_H_
#include <stdint.h>

#include "common.h"
#include "core.h"
#include "instrumentation.h"
#include "actuation_logic.h"

// channel -> sensor # -> val
extern uint32_t sensors[2][2];
// channel -> sensor # -> demux output # -> val
extern uint32_t sensors_demux[2][2][2];

extern uint8_t trip_signals[NTRIP][4];
extern struct instrumentation_command inst_command_buf[4];

extern uint8_t actuator_state[NDEV];
extern uint8_t device_actuation_logic[2][NDEV];
extern struct actuation_command *act_command_buf[2];

//EI mode:
// mode = 0 => no error
// mode = 1 => error
// mode = 2 => nondet error
extern uint8_t error_instrumentation_mode[NINSTR];
extern uint8_t error_instrumentation[NINSTR];
// ES ch mode:
// mode = 0 => no error
// mode = 1 => demux error (out 0)
// mode = 2 => demux error (out 1)
// mode = 3 => sensor error (error in both demux outs)
```

```
// mode = 4 => nondet sensor error
// mode = 5 => nondet demux error
extern uint8_t error_sensor_mode[2][2];
extern uint8_t error_sensor[2][2];
extern uint8_t error_sensor_demux[2][2][2];

#ifdef DEBUG
#define DEBUG_PRINTF(X) printf X
#else
#define DEBUG_PRINTF(X)
#endif

#ifdef PLATFORM_HOST
#include <assert.h>
#define ASSERT(x) assert(x)
#else
#define ASSERT(x)
#endif // PLATFORM_HOST

#if defined(PLATFORM_HOST) && defined(USE_PTHREADS)
#include <pthread.h>
extern pthread_mutex_t display_mutex;
extern pthread_mutex_t mem_mutex;
#define MUTEX_LOCK(x) pthread_mutex_lock(x)
#define MUTEX_UNLOCK(x) pthread_mutex_unlock(x)
#else
#define MUTEX_LOCK(x)
#define MUTEX_UNLOCK(x)
#endif // defined(PLATFORM_HOST) && defined(USE_PTHREADS)

////////////////////////////////////////
// Reading signals and values //
////////////////////////////////////////

/*@requires \valid(val);
  @requires div < NINSTR;
  @requires channel < NTRIP;
  @assigns *val;
  @ensures -1 <= \result <= 0;
  @ensures \result == 0 ==> *val <= 0x80000000;
 */
int read_instrumentation_channel(uint8_t div, uint8_t channel, uint32_t *val);

int get_instrumentation_value(uint8_t division, uint8_t ch, uint32_t *value);
int get_instrumentation_trip(uint8_t division, uint8_t ch, uint8_t *value);
int get_instrumentation_mode(uint8_t division, uint8_t ch, uint8_t *value);
int get_instrumentation_maintenance(uint8_t division, uint8_t *value);

// Reading actuation signals
/*@ requires i <= 1;
  @ requires device < NDEV;
  @ requires \valid(value);
  @ assigns *value;
  @ ensures (\result == 0) ==> (*value == 0 || *value == 1);
  @ ensures (\result != 0) ==> (*value == \old(*value));
 */
int get_actuation_state(uint8_t i, uint8_t device, uint8_t *value);

/*@requires \valid(&arr[0.. NTRIP-1][0.. NINSTR-1]);
  @assigns *(arr[0.. NTRIP-1]+(0.. NINSTR-1));
 */
int read_instrumentation_trip_signals(uint8_t arr[3][4]);

////////////////////////////////////////
// Setting output signals //
////////////////////////////////////////
```

```
99  |
100 | int reset_actuation_logic(uint8_t logic_no, uint8_t device_no, uint8_t reset_val);
101 |
102 | /*@requires logic_no < NVOTE_LOGIC;
103 |   @requires device_no < NDEV;
104 |   @assigns \nothing; // Not entirely true, but we'll never mention that state
105 |   @ensures -1 <= \result <= 0;
106 |  */
107 | int set_output_actuation_logic(uint8_t logic_no, uint8_t device_no, uint8_t on);
108 |
109 | /*@requires division < NINSTR;
110 |   @requires channel < NTRIP;
111 |   @assigns \nothing; // Not entirely true, but we'll never mention that state
112 | */
113 | int set_output_instrumentation_trip(uint8_t division, uint8_t channel, uint8_t val);
114 |
115 | /*@ requires device_no <= 1;
116 |   @ assigns \nothing;
117 | */
118 | int set_actuate_device(uint8_t device_no, uint8_t on);
119 |
120 | ////////////////////////////////////////
121 | // Sending commands between components //
122 | ////////////////////////////////////////
123 | /**
124 |  * Read RTS command from the user
125 |  * Platform specific
126 |  */
127 | int read_rts_command(struct rts_command *cmd);
128 |
129 | /* Communicate with instrumentation division */
130 |
131 | /*@requires division < NINSTR;
132 |   @requires \valid(cmd);
133 |   @assigns cmd->type, cmd->cmd;
134 |   @ensures -1 <= \result <= 1;
135 | */
136 | int read_instrumentation_command(uint8_t division, struct instrumentation_command *cmd);
137 |
138 | /*@requires division < NINSTR;
139 |   @requires \valid(cmd);
140 |   @assigns \nothing; // not entirely true, but we'll never mention that state
141 |   @ensures -1 <= \result <= 0;
142 | */
143 | int send_instrumentation_command(uint8_t division, struct instrumentation_command *cmd);
144 |
145 | /**
146 |  * Read external command, setting *cmd. Does not block.
147 |  * Platform specific
148 |  */
149 | /*@requires \valid(cmd);
150 |   @assigns cmd->on;
151 |   @assigns cmd->device;
152 |   @ensures -1 <= \result <= 1;
153 |  */
154 | int read_actuation_command(uint8_t id, struct actuation_command *cmd);
155 |
156 | /**
157 |  * Physically set actuator to a new value
158 |  * Platform specific
159 |  */
160 | int send_actuation_command(uint8_t actuator,
161 |                            struct actuation_command *cmd);
162 |
163 |
164 | ////////////////////////////////////////////
```

```
// Self Test state //
/////////////////////////////////////////

/*@ assigns \nothing; */
uint8_t is_test_running(void);

/*@ assigns \nothing; */
void set_test_running(int val);

/*@ assigns \nothing;
  @ ensures \result < NDEV;
*/
uint8_t get_test_device(void);

/*@ requires \valid(id) && \valid(&id[1]);
  @ assigns id[0], id[1];
  @ ensures id[0] < NINSTR;
  @ ensures id[1] < NINSTR;
*/
void get_test_instrumentation(uint8_t *id);

/*@ requires \valid(setpoints + (0.. NTRIP-1));
  @ requires id < NINSTR;
  @ assigns setpoints[0.. NTRIP-1];
  @ ensures -1 <= \result <= 0;
*/
int get_instrumentation_test_setpoints(uint8_t id, uint32_t *setpoints);

/*@ requires div < NINSTR;
  @ assigns core.test.test_instrumentation_done[div];
  @ ensures core.test.test_instrumentation_done[div] == v;
*/
void set_instrumentation_test_complete(uint8_t div, int v);

/*@ requires id < NINSTR;
  @ assigns \nothing;
*/
int is_instrumentation_test_complete(uint8_t id);

/*@ requires div < NINSTR;
  @ requires channel < NTRIP;
  @ requires \valid(val);
  @ assigns *val;
  @ ensures -1 <= \result <= 0;
*/
int read_test_instrumentation_channel(uint8_t div, uint8_t channel, uint32_t *val);

/*@ assigns \nothing;
  @ ensures \result < NVOTE_LOGIC;
*/
uint8_t get_test_actuation_unit(void);

// NOTE: this is actually never used (only in `bottom.c`)
int is_actuation_unit_under_test(uint8_t id);

/*@ requires div < NVOTE_LOGIC;
  @ assigns core.test.test_actuation_unit_done[div];
  @ ensures core.test.test_actuation_unit_done[div] == v;
*/
void set_actuation_unit_test_complete(uint8_t div, int v);

/*@ requires id < NVOTE_LOGIC;
  @ assigns core.test.actuation_old_vote;
  @ ensures core.test.actuation_old_vote == v;
*/
void set_actuation_unit_test_input_vote(uint8_t id, int v);
```

```
231
232 /*@ requires id < NVOTE_LOGIC;
233   @ assigns \nothing;
234 */
235 int is_actuation_unit_test_complete(uint8_t id);
236
237 /*@ requires dev < NDEV;
238   @ assigns core.test.test_device_result[dev];
239   @ ensures core.test.test_device_result[dev] == result;
240 */
241 void set_actuate_test_result(uint8_t dev, uint8_t result);
242
243 /*@ requires dev < NDEV;
244   @ assigns core.test.test_device_done[dev];
245   @ ensures core.test.test_device_done[dev] == v;
246 */
247 void set_actuate_test_complete(uint8_t dev, int v);
248
249 /*@ requires dev < NDEV;
250   @ assigns \nothing;
251 */
252 int is_actuate_test_complete(uint8_t dev);
253
254 ////////////////////////////////////////////
255 // General Utilities //
256 ////////////////////////////////////////////
257
258 /**
259  * Return uptime in seconds
260  * Platform specific
261  */
262 uint32_t time_in_s(void);
263
264 /**
265  * Update user display
266  * Platform specific
267  */
268 void update_display(void);
269
270 /**
271  * Poll sensors for new values
272  * Platform specific
273  */
274 void update_sensors(void);
275
276 #endif // PLATFORM_H_
```

Listing E.38: Listing Interface of RTS.

```
1  #ifndef RTS_H_
2  #define RTS_H_
3  #include <stdint.h>
4
5  #include "instrumentation.h"
6  #include "actuation_logic.h"
7  #include "actuate.h"
8
9  typedef uint32_t sensor_reading_t;
10
11 #endif // RTS_H_
```

Listing E.39: Listing Interface Model of sense_actuate.

```
1 #ifndef SENSE_ACTUATE_H_
```

```
#define SENSE_ACTUATE_H_

#include "common.h"
#include "instrumentation.h"
#include "actuation_logic.h"

/* Initialize state for core `core_id`.
 * requires instrumentation is an array of NINSTRUMENTATION/NCORE_ID instrumentation structs
 * requires actuation_logic is an array of NACTUATION_LOGIC/NCORE_ID actuation_logic structs
 * returns < 0 on error
 */
int sense_actuate_init(int core_id,
                       struct instrumentation_state *instrumentation,
                       struct actuation_logic *actuation);

/* Advance state for core `core_id`.
 * requires instrumentation is an array of NINSTRUMENTATION/NCORE_ID instrumentation structs
 * requires actuation_logic is an array of NACTUATION_LOGIC/NCORE_ID actuation_logic structs
 * returns < 0 on error
 */
int sense_actuate_step_0(struct instrumentation_state *instrumentation,
                         struct actuation_logic *actuation);
int sense_actuate_step_1(struct instrumentation_state *instrumentation,
                         struct actuation_logic *actuation);

#endif // SENSE_ACTUATE_H_
```

Appendix F

Software Implementation

Listing F.1: C implementation of rv32_main.

```
/**
 * Main program entry for RTS
 */
// System includes
#include <stdint.h>
#include <stdlib.h>
#include <string.h>

// Board includes
#include "bsp.h"
#include "printf.h"

// RTS includes
#include "common.h"
#include "core.h"
#include "instrumentation.h"
#include "actuation_logic.h"
#include "sense_actuate.h"
#include "platform.h"
#include "actuation_logic.h"

extern struct instrumentation_state instrumentation[4];

void update_display()
{
#if CLEAR_SCREEN
  // This starts printing from the top of the screen
  printf("\e[s\e[1;1H");//\e[2J");
#endif
  for (int line = 0; line < NLINES; ++line) {
#if CLEAR_SCREEN
    printf("\e[0K");
#endif
    printf("%s%s", core.ui.display[line], line == NLINES-1 ? "" : "\n");
  }
#if CLEAR_SCREEN
  printf("\e[u");
#endif
}

int read_rts_command(struct rts_command *cmd)
{
```

```
  int ok = 0;
#ifndef ENABLE_SELF_TEST
  const char delimiter[2] = " ";
  char line[254] = {0};
  char *token = NULL;
  int linelen = 0;

  printf("\nEnter command and press enter:\n");
  memset(line,0,sizeof(line));
  for (unsigned int i = 0; i < sizeof(line); i++) {
    line[i] = soc_getchar();
    linelen = i;
    if (line[i] == 0 || line[i] == '\n') {
      break;
    }
  }
  printf(">>>%s<<<[%d]\n",line, linelen);

#if CLEAR_SCREEN
  printf("\e[%d;1H\e[2K> ", NLINES+1);
#endif

  if (linelen < 4) {
    // Too short to be a valid command. "A 1 1\n" is the shortest command
    return ok;
  }

  /* get the first token */
  printf("About to call strtok\n");
  token = strtok(line, delimiter);
  printf("strtok called\n");

  if (token != NULL) {
    printf("Command = %s\n",token);
    switch (token[0]) {
      case 'A':
        cmd->type = ACTUATION_COMMAND;
        // "A %hhd %hhd", &device, &on
        token = strtok(NULL, delimiter);
        if (token != NULL) {
          printf("cmd->cmd.act.device = %s\n",token);
          cmd->cmd.act.device = (uint8_t)atoi(token);
          token = strtok(NULL, delimiter);
          if (token != NULL) {
            printf("cmd->cmd.act.on = %s\n",token);
            cmd->cmd.act.on = (uint8_t)atoi(token);
            printf("ACTUATION_COMMAND dev=%u on=%u\n",
                  cmd->cmd.act.device, cmd->cmd.act.on);
            ok = 1;
          }
        }
        break;
      case 'M':
        cmd->type = INSTRUMENTATION_COMMAND;
        cmd->cmd.instrumentation.type = SET_MAINTENANCE;
        // "M %hhd %hhd", &div, &on
        token = strtok(NULL, delimiter);
        if (token != NULL) {
          printf("cmd->instrumentation_division = %s\n",token);
          cmd->instrumentation_division = (uint8_t)atoi(token);
          token = strtok(NULL, delimiter);
          if (token != NULL) {
            printf("cmd->cmd.instrumentation.cmd.maintenance.on = %s\n",token);
            cmd->cmd.instrumentation.cmd.maintenance.on = (uint8_t)atoi(token);
            printf("INSTRUMENTATION_COMMAND MAINTENANCE div=%u on=%u\n",
                  cmd->instrumentation_division,
```

```
                cmd->cmd.instrumentation.cmd.maintenance.on);
          ok = 1;
        }
      }
      break;
    case 'B':
      cmd->type = INSTRUMENTATION_COMMAND;
      cmd->cmd.instrumentation.type = SET_MODE;
      // "B %hhd %hhd %hhd", &div, &ch, &mode
      token = strtok(NULL, delimiter);
      if (token != NULL) {
        printf("cmd->instrumentation_division = %s\n",token);
        cmd->instrumentation_division = (uint8_t)atoi(token);
        token = strtok(NULL, delimiter);
        if (token != NULL) {
          printf("cmd->cmd.instrumentation.cmd.mode.channel = %s\n",token);
          cmd->cmd.instrumentation.cmd.mode.channel = (uint8_t)atoi(token);
          token = strtok(NULL, delimiter);
          if (token != NULL) {
            printf("cmd->cmd.instrumentation.cmd.mode.mode_val = %s\n",token);
            cmd->cmd.instrumentation.cmd.mode.mode_val = (uint8_t)atoi(token);
            printf("INSTRUMENTATION_COMMAND MODE div=%u channel=%u mode=%u\n",
                cmd->instrumentation_division,
                cmd->cmd.instrumentation.cmd.mode.channel,
                cmd->cmd.instrumentation.cmd.mode.mode_val);
            ok = 1;
          }
        }
      }
      break;
    case 'S':
      cmd->type = INSTRUMENTATION_COMMAND;
      cmd->cmd.instrumentation.type = SET_SETPOINT;
      // "S %hhd %hhd %d", &div, &ch, &val
      token = strtok(NULL, delimiter);
      if ((token != NULL) && (token[0] != '\n')) {
        printf("cmd->instrumentation_division = %s\n",token);
        cmd->instrumentation_division = (uint8_t)atoi(token);
        token = strtok(NULL, delimiter);
        if ((token != NULL) && (token[0] != '\n')) {
          printf("cmd->cmd.instrumentation.cmd.setpoint.channel = %s\n",token);
          cmd->cmd.instrumentation.cmd.setpoint.channel = (uint8_t)atoi(token);
          token = strtok(NULL, delimiter);
          if ((token != NULL) && (token[0] != '\n')) {
            printf("cmd->cmd.instrumentation.cmd.setpoint.val = %s\n",token);
            cmd->cmd.instrumentation.cmd.setpoint.val = (uint32_t)atoi(token);
            printf("INSTRUMENTATION_COMMAND SETPOINT div=%u channel=%u val=%u\n",
                cmd->instrumentation_division,
                cmd->cmd.instrumentation.cmd.setpoint.channel,
                cmd->cmd.instrumentation.cmd.setpoint.val);
            ok = 1;
          }
        }
      }
      break;
    default:
      break;
   }
  }
#endif /* #ifndef ENABLE_SELF_TEST */
  return ok;
}

uint32_t get_sensor_data(uint8_t sensor_addr)
{
  uint32_t data = 0;
```

```
175   uint32_t addr = 0;
176   uint32_t result = 0;
177   uint8_t intermidiate = 0;
178
179   // run 4 times to get all 32bits of uint32_t value
180   for (uint8_t i = 0; i < 4; i++)
181   {
182     data = i;
183     // Set data pointer reg
184     write_reg(I2C_REG_DATA, data);
185     // Set write addr
186     addr = (sensor_addr << 1) | 0x1;
187     write_reg(I2C_REG_ADDR, addr);
188     // Wait for transaction to finish
189     while (read_reg(I2C_REG_STATUS) != 1) {
190       ;;
191     }
192     // Set read addr
193     addr = (sensor_addr << 1);
194     write_reg(I2C_REG_ADDR, addr);
195     // Wait for transaction to finish
196     while (read_reg(I2C_REG_STATUS) != 1) {
197       ;;
198     }
199     // Update the result
200     intermidiate = read_reg(I2C_REG_DATA);
201     result = result | ( intermidiate << i*8);
202   }
203   return result;
204 }
205
206 void update_sensors(void)
207 {
208   uint32_t val0, val1;
209   val0 = get_sensor_data(TEMP_0_I2C_ADDR);
210   val1 = get_sensor_data(TEMP_1_I2C_ADDR);
211   sensors_demux[0][T][0] = val0;
212   sensors_demux[0][T][1] = val1;
213   sensors_demux[1][T][0] = val0;
214   sensors_demux[1][T][1] = val1;
215
216   val0 = get_sensor_data(PRESSURE_0_I2C_ADDR);
217   val1 = get_sensor_data(PRESSURE_1_I2C_ADDR);
218   sensors_demux[0][P][0] = val0;
219   sensors_demux[0][P][1] = val1;
220   sensors_demux[1][P][0] = val0;
221   sensors_demux[1][P][1] = val1;
222 }
223
224 /**
225  * Read external command, setting *cmd. Does not block.
226  * Platform specific
227  */
228 int read_actuation_command(uint8_t id, struct actuation_command *cmd) {
229   cmd->device = id;
230   cmd->on = (uint8_t) (read_reg(GPIO_REG) & (id+1));
231   DEBUG_PRINTF(("<main.c> read_actuation_command: cmd->device=%u, cmd->on=%u\n",cmd->device,
        ↪ cmd->on));
232   return 1;
233 }
234
235 /**
236  * Physically set actuator to a new value
237  * Platform specific
238  */
239 int send_actuation_command(uint8_t id, struct actuation_command *cmd)
```

```
{
  DEBUG_PRINTF(("<main.c> send_actuation_command, id=%u, cmd->device=%u, cmd->on=%u\n",id,
      ↪ cmd->device,cmd->on));
  if ((id < 2) && (cmd->on < 2) ) {
    uint32_t gpio_val = read_reg(GPIO_REG);
    if (id == 0) {
      // Set the actuator bit to zero
      gpio_val = gpio_val & 0xFFFFFFFE;
      // Se the actuator bit to cmd->on value
      gpio_val = gpio_val | cmd->on;
    }
    else {
      // id == 1
      // Set the actuator bit to zero
      gpio_val = gpio_val & 0xFFFFFFFD;
      // Se the actuator bit to cmd->on value
      // Bit shift by one left
      gpio_val = gpio_val | (cmd->on << 1);
    }
    write_reg(GPIO_REG, gpio_val);
    return 0;
  }
  return -1;
}

int main(void)
{
#if CLEAR_SCREEN
  // Prep the screen
  printf("\e[1;1H\e[2J");
  printf("\e[%d;3H\e[2K> ", NLINES+1);
#endif

  //struct rts_command cmd;// = (struct rts_command *)malloc(sizeof(*cmd));
  core_init(&core);
  sense_actuate_init(0, &instrumentation[0], &actuation_logic[0]);
  sense_actuate_init(1, &instrumentation[0], &actuation_logic[0]);

  char line[256];
  while (1)
  {
    //read_rts_command(&cmd);

    update_sensors();
    sprintf(line, "HW ACTUATORS 0x%X", read_reg(GPIO_REG));
    set_display_line(&core.ui, 8, line, 0);
    int retval = core_step(&core);
    DEBUG_PRINTF(("<main.c> core_step= 0x%X\n",retval));
    char line[256];
    sprintf(line, "Uptime: [%u]s\n",time_in_s());
    set_display_line(&core.ui, 9, line, 0);
    update_display();
    sense_actuate_step_0(&instrumentation[0], &actuation_logic[0]);
    sense_actuate_step_1(&instrumentation[2], &actuation_logic[1]);
  }

  return 0;
}
```

Listing F.2: C implementation of common.

```
#include <stddef.h>
#include <stdint.h>
#include <stdlib.h>

#include "platform.h"
```

```c
#include "common.h"
#include "core.h"
#include "instrumentation.h"
#include "actuation_logic.h"
#include "sense_actuate.h"

#ifdef PLATFORM_HOST
#include <stdio.h>
#else
#include "printf.h"
#endif

struct core_state core = {0};
struct instrumentation_state instrumentation[4];
struct actuation_logic actuation_logic[2];

// channel -> sensor # -> val
uint32_t sensors[2][2];
// channel -> sensor # -> demux output # -> val
uint32_t sensors_demux[2][2][2];

uint8_t trip_signals[NTRIP][4];
struct instrumentation_command inst_command_buf[4];

uint8_t actuator_state[NDEV];
uint8_t device_actuation_logic[2][NDEV];

//EI mode:
// mode = 0 => no error
// mode = 1 => error
// mode = 2 => nondet error
uint8_t error_instrumentation_mode[NINSTR];
uint8_t error_instrumentation[NINSTR];
// ES ch mode:
// mode = 0 => no error
// mode = 1 => demux error (out 0)
// mode = 2 => demux error (out 1)
// mode = 3 => sensor error (error in both demux outs)
// mode = 4 => nondet sensor error
// mode = 5 => nondet demux error
uint8_t error_sensor_mode[2][2];
uint8_t error_sensor[2][2];
uint8_t error_sensor_demux[2][2][2];

int read_instrumentation_channel(uint8_t div, uint8_t channel, uint32_t *val) {
  MUTEX_LOCK(&mem_mutex);
  int sensor = div/2;
  int demux_out = div%2;
  *val = sensors_demux[channel][sensor][demux_out];
  MUTEX_UNLOCK(&mem_mutex);
  DEBUG_PRINTF(("<common.c> read_instrumentation_channel: div=%u,channel=%u,val=%u\n",div,
      ↪ channel,*val));
  return 0;
}

int get_instrumentation_value(uint8_t division, uint8_t ch, uint32_t *value) {
  MUTEX_LOCK(&mem_mutex);
  if (!error_instrumentation[division])
    *value = instrumentation[division].reading[ch];
  MUTEX_UNLOCK(&mem_mutex);
  DEBUG_PRINTF(("<common.c> get_instrumentation_value: error=%u, division=%u,ch=%u,val=%u\n"
      ↪ ,error_instrumentation[division], division,ch,*value));
  return 0;
}

int get_instrumentation_trip(uint8_t division, uint8_t ch, uint8_t *value) {
```

```
70    MUTEX_LOCK(&mem_mutex);
71    if (!error_instrumentation[division])
72      *value = instrumentation[division].sensor_trip[ch];
73    MUTEX_UNLOCK(&mem_mutex);
74    DEBUG_PRINTF(("<common.c> get_instrumentation_trip: error=%u, division=%u,ch=%u,val=%u\n",
          ↪ error_instrumentation[division], division,ch,*value));
75    return 0;
76  }
77
78  int get_instrumentation_mode(uint8_t division, uint8_t ch, uint8_t *value) {
79    MUTEX_LOCK(&mem_mutex);
80    if (!error_instrumentation[division])
81      *value = instrumentation[division].mode[ch];
82    MUTEX_UNLOCK(&mem_mutex);
83    DEBUG_PRINTF(("<common.c> get_instrumentation_mode: error=%u, division=%u,ch=%u,val=%u\n",
          ↪ error_instrumentation[division], division,ch,*value));
84    return 0;
85  }
86
87  int get_instrumentation_maintenance(uint8_t division, uint8_t *value) {
88    MUTEX_LOCK(&mem_mutex);
89    if (!error_instrumentation[division])
90      *value = instrumentation[division].maintenance;
91    MUTEX_UNLOCK(&mem_mutex);
92    DEBUG_PRINTF(("<common.c> get_instrumentation_maintenance: error=%u, division=%u,val=%u\n"
          ↪ ,error_instrumentation[division],division,*value));
93    return 0;
94  }
95
96  int get_actuation_state(uint8_t i, uint8_t device, uint8_t *value) {
97    MUTEX_LOCK(&mem_mutex);
98    *value = device_actuation_logic[i][device];
99    MUTEX_UNLOCK(&mem_mutex);
100   DEBUG_PRINTF(("<common.c> get_actuation_state: i=%u,device=%u,val=%u\n",i,device,*value));
101   return 0;
102 }
103
104 int read_instrumentation_trip_signals(uint8_t arr[3][4]) {
105   DEBUG_PRINTF(("<common.c> read_instrumentation_trip_signals: ["));
106   for (int i = 0; i < NTRIP; ++i) {
107     DEBUG_PRINTF(("["));
108     for (int div = 0; div < 4; ++div) {
109       MUTEX_LOCK(&mem_mutex);
110       arr[i][div] = trip_signals[i][div];
111       DEBUG_PRINTF(("%u",trip_signals[i][div]));
112       MUTEX_UNLOCK(&mem_mutex);
113     }
114     DEBUG_PRINTF(("],"));
115   }
116   DEBUG_PRINTF(("]\n"));
117   return 0;
118 }
119
120 int reset_actuation_logic(uint8_t logic_no, uint8_t device_no, uint8_t reset_val) {
121   MUTEX_LOCK(&mem_mutex);
122   actuation_logic[logic_no].vote_actuate[device_no] = reset_val;
123   MUTEX_UNLOCK(&mem_mutex);
124   DEBUG_PRINTF(("<common.c> reset_actuation_logic: logic_no=%u,device=%u,reset_val=%u\n",
          ↪ logic_no,device_no,reset_val));
125   return 0;
126 }
127
128 int set_output_actuation_logic(uint8_t logic_no, uint8_t device_no, uint8_t on) {
129   ASSERT(logic_no < 2);
130   ASSERT(device_no < 2);
131
```

```
132   MUTEX_LOCK(&mem_mutex);
133   device_actuation_logic[logic_no][device_no] = on;
134   MUTEX_UNLOCK(&mem_mutex);
135   DEBUG_PRINTF(("<common.c> set_output_actuation_logic: logic_no=%u,device=%u,on=%u\n",
          ↪ logic_no,device_no,on));
136   return 0;
137 }
138
139 int set_output_instrumentation_trip(uint8_t div, uint8_t channel, uint8_t val) {
140   MUTEX_LOCK(&mem_mutex);
141   if (!error_instrumentation[div])
142     trip_signals[channel][div] = val;
143   MUTEX_UNLOCK(&mem_mutex);
144   DEBUG_PRINTF(("<common.c> set_output_instrumentation_trip: error=%u,div=%u,channel=%u,val
          ↪ =%u\n",error_instrumentation[div], div, channel, val));
145   return 0;
146 }
147
148 int set_actuate_device(uint8_t device_no, uint8_t on)
149 {
150   MUTEX_LOCK(&mem_mutex);
151   actuator_state[device_no] = on;
152   MUTEX_UNLOCK(&mem_mutex);
153   DEBUG_PRINTF(("<common.c> set_actuate_device: dev %u, on %u\n",device_no, on));
154   return 0;
155 }
156
157 int read_instrumentation_command(uint8_t div,
158                                  struct instrumentation_command *cmd) {
159   DEBUG_PRINTF(("<common.c> read_instrumentation_command\n"));
160   if ((div < 4) && (inst_command_buf[div].valid == 1)) {
161     cmd->type = inst_command_buf[div].type;
162     cmd->cmd = inst_command_buf[div].cmd;
163     inst_command_buf[div].valid = 0;
164     return 1;
165   }
166   return 0;
167 }
168
169 int send_instrumentation_command(uint8_t div,
170                                  struct instrumentation_command *cmd) {
171   DEBUG_PRINTF(("<common.c> send_instrumentation_command\n"));
172   if (div < 4) {
173     inst_command_buf[div].type = cmd->type;
174     inst_command_buf[div].cmd = cmd->cmd;
175     inst_command_buf[div].valid = 1;
176     return 0;
177   }
178   return -1;
179 }
180
181 uint8_t is_test_running()
182 {
183   MUTEX_LOCK(&mem_mutex);
184   uint8_t ret = core.test.self_test_running;
185   MUTEX_UNLOCK(&mem_mutex);
186   DEBUG_PRINTF(("<common.c> is_test_running? %u\n",ret));
187   return ret;
188 }
189
190 void set_test_running(int val)
191 {
192   MUTEX_LOCK(&mem_mutex);
193   core.test.self_test_running = val;
194   MUTEX_UNLOCK(&mem_mutex);
195   DEBUG_PRINTF(("<common.c> set_test_running: %i\n",core.test.self_test_running));
```

```
196 }
197
198 uint8_t get_test_device()
199 {
200   DEBUG_PRINTF(("<common.c> get_test_device: %u\n",core.test.test_device));
201   return core.test.test_device;
202 }
203
204 void get_test_instrumentation(uint8_t *id)
205 {
206   id[0] = core.test.test_instrumentation[0];
207   id[1] = core.test.test_instrumentation[1];
208   DEBUG_PRINTF(("<common.c> get_test_instrumentation\n"));
209 }
210
211 int get_instrumentation_test_setpoints(uint8_t id, uint32_t *setpoints)
212 {
213   setpoints[0] = core.test.test_setpoints[id][0];
214   setpoints[1] = core.test.test_setpoints[id][1];
215   setpoints[2] = core.test.test_setpoints[id][2];
216   DEBUG_PRINTF(("<common.c> get_instrumentation_test_setpoints\n"));
217   return 0;
218 }
219
220 void set_instrumentation_test_complete(uint8_t div, int v)
221 {
222   MUTEX_LOCK(&mem_mutex);
223   core.test.test_instrumentation_done[div] = v;
224   MUTEX_UNLOCK(&mem_mutex);
225   DEBUG_PRINTF(("<common.c> set_instrumentation_test_complete: div=%u,v=%i\n",div,v));
226 }
227
228 int is_instrumentation_test_complete(uint8_t id)
229 {
230   MUTEX_LOCK(&mem_mutex);
231   int ret = core.test.test_instrumentation_done[id];
232   MUTEX_UNLOCK(&mem_mutex);
233   DEBUG_PRINTF(("<common.c> is_instrumentation_test_complete: id=%u,ret=%i\n",id,ret));
234   return ret;
235 }
236
237 int read_test_instrumentation_channel(uint8_t div, uint8_t channel, uint32_t *val)
238 {
239   MUTEX_LOCK(&mem_mutex);
240   *val = core.test.test_inputs[div][channel];
241   MUTEX_UNLOCK(&mem_mutex);
242   DEBUG_PRINTF(("<common.c> read_test_instrumentation_channel: div=%u,channel=%u,val=%u\n",
        ↪ div,channel,*val));
243   return 0;
244 }
245
246 uint8_t get_test_actuation_unit()
247 {
248   MUTEX_LOCK(&mem_mutex);
249   uint8_t ret = core.test.test_actuation_unit;
250   MUTEX_UNLOCK(&mem_mutex);
251   DEBUG_PRINTF(("<common.c> get_test_actuation_unit: %u\n",ret));
252   return ret;
253 }
254
255 void set_actuation_unit_test_complete(uint8_t div, int v)
256 {
257   MUTEX_LOCK(&mem_mutex);
258   core.test.test_actuation_unit_done[div] = v;
259   MUTEX_UNLOCK(&mem_mutex);
260   DEBUG_PRINTF(("<common.c> set_actuation_unit_test_complete: div %u, v=%i\n",div,v));
```

```
}

void set_actuation_unit_test_input_vote(uint8_t id, int v)
{
  MUTEX_LOCK(&mem_mutex);
  core.test.actuation_old_vote = v != 0;
  MUTEX_UNLOCK(&mem_mutex);
  DEBUG_PRINTF(("<common.c> set_actuation_unit_test_input_vote: id %u, v=%i\n",id,v));
}

int is_actuation_unit_test_complete(uint8_t id)
{
  MUTEX_LOCK(&mem_mutex);
  int ret = core.test.test_actuation_unit_done[id];
  MUTEX_UNLOCK(&mem_mutex);
  DEBUG_PRINTF(("<common.c> is_actuation_unit_test_complete: %i\n",ret));
  return ret;
}

void set_actuate_test_result(uint8_t dev, uint8_t result)
{
  MUTEX_LOCK(&mem_mutex);
  core.test.test_device_result[dev] = result;
  MUTEX_UNLOCK(&mem_mutex);
  DEBUG_PRINTF(("<common.c> set_actuate_test_result: dev %u, result=%u\n",dev,result));
}

void set_actuate_test_complete(uint8_t dev, int v)
{
  MUTEX_LOCK(&mem_mutex);
  core.test.test_device_done[dev] = v;
  MUTEX_UNLOCK(&mem_mutex);
  DEBUG_PRINTF(("<common.c> set_actuate_test_complete: dev %u, v=%i\n",dev,v));
}

int is_actuate_test_complete(uint8_t dev)
{
  MUTEX_LOCK(&mem_mutex);
  int ret = core.test.test_device_done[dev];
  MUTEX_UNLOCK(&mem_mutex);
  DEBUG_PRINTF(("<common.c> is_actuate_test_complete: %i\n",ret));
  return ret;
}
```

Listing F.3: C implementation of posix_main.

```
#include "common.h"
#include "core.h"
#include "instrumentation.h"
#include "actuation_logic.h"
#include "sense_actuate.h"
#include "platform.h"

#include <poll.h>
#include <fcntl.h>
#include <stdio.h>
#include <termios.h>
#include <unistd.h>
#include <stddef.h>
#include <stdint.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/select.h>
#include <time.h>

```

```
extern struct instrumentation_state instrumentation[4];
struct actuation_command *act_command_buf[2];

#define min(_a, _b) ((_a) < (_b) ? (_a) : (_b))
#define max(_a, _b) ((_a) > (_b) ? (_a) : (_b))

#include <pthread.h>
pthread_mutex_t display_mutex = PTHREAD_MUTEX_INITIALIZER;
pthread_mutex_t mem_mutex = PTHREAD_MUTEX_INITIALIZER;

#ifndef T0
#define T0 200
#endif

#ifndef P0
#define P0 1152600
#endif

// Bias to simulated sensor readings in degrees F
#ifndef T_BIAS
#define T_BIAS 0
#endif

// Bias to simulated sensor readings in 10^-5 lb/in2
#ifndef P_BIAS
#define P_BIAS 0
#endif

#ifndef SENSOR_UPDATE_MS
#define SENSOR_UPDATE_MS 500
#endif

int clear_screen() {
  return (isatty(fileno(stdin)) && (NULL == getenv("RTS_NOCLEAR")));
}

void update_display() {
  if (clear_screen()) {
    printf("\e[s\e[1;1H");//\e[2J");
  }
  for (int line = 0; line < NLINES; ++line) {
    printf("\e[0K");
    printf("%s%s", core.ui.display[line], line == NLINES-1 ? "" : "\n");
  }
  if (clear_screen()) {
    printf("\e[u");
  }
}

int read_rts_command(struct rts_command *cmd) {
  int ok = 0;
  uint8_t device, on, div, ch, mode, sensor;
  uint32_t val;
  char *line = NULL;
  size_t linecap = 0;
  ssize_t linelen;

  /* if (isatty(fileno(stdin))) { */
  /* set_display_line(&ui, 9, (char *)"> ", 0); */
  /* } */
  struct pollfd fds;
  fds.fd = STDIN_FILENO;
  fds.events = POLLIN;
  fds.revents = POLLIN;
  if(poll(&fds, 1, 100) < 1) {
    return 0;
```

```
  }
  linelen = getline(&line, &linecap, stdin);

  if (linelen == EOF)
    exit(0);

  if (linelen < 0)
    return 0;

  MUTEX_LOCK(&display_mutex);

  if (clear_screen()) {
    printf("\e[%d;1H\e[2K> ", NLINES+1);
  }

  MUTEX_UNLOCK(&display_mutex);

  if (2 == (ok = sscanf(line, "A %hhd %hhd", &device, &on))) {
    cmd->type = ACTUATION_COMMAND;
    cmd->cmd.act.device = device;
    cmd->cmd.act.on = on;
    DEBUG_PRINTF(("<main.c> read_rts_command ACTUATION_COMMAND dev=%u on=%u\n",
                  cmd->cmd.act.device, cmd->cmd.act.on));
    ok = 1;
  } else if (2 == (ok = sscanf(line, "M %hhd %hhd", &div, &on))) {
    cmd->type = INSTRUMENTATION_COMMAND;
    cmd->instrumentation_division = div;
    cmd->cmd.instrumentation.type = SET_MAINTENANCE;
    cmd->cmd.instrumentation.cmd.maintenance.on = on;
    DEBUG_PRINTF(("<main.c> read_rts_command INSTRUMENTATION_COMMAND MAINTENANCE div=%u on=%u
        ↪ , type=%u\n",
                  cmd->instrumentation_division,
                  cmd->cmd.instrumentation.cmd.maintenance.on,
                  cmd->cmd.instrumentation.type));
    ASSERT(on == 0 || on == 1);
    ok = 1;
  } else if (3 == (ok = sscanf(line, "B %hhd %hhd %hhd", &div, &ch, &mode))) {
    cmd->type = INSTRUMENTATION_COMMAND;
    cmd->instrumentation_division = div;
    cmd->cmd.instrumentation.type = SET_MODE;
    cmd->cmd.instrumentation.cmd.mode.channel = ch;
    cmd->cmd.instrumentation.cmd.mode.mode_val = mode;
    DEBUG_PRINTF(("<main.c> read_rts_command INSTRUMENTATION_COMMAND MODE div=%u channel=%u
        ↪ mode=%u type=%u\n",
                  cmd->instrumentation_division,
                  cmd->cmd.instrumentation.cmd.mode.channel,
                  cmd->cmd.instrumentation.cmd.mode.mode_val,
                  cmd->cmd.instrumentation.type));
    ok = 1;
  } else if (3 == (ok = sscanf(line, "S %hhd %hhd %d", &div, &ch, &val))) {
    cmd->type = INSTRUMENTATION_COMMAND;
    cmd->instrumentation_division = div;
    cmd->cmd.instrumentation.type = SET_SETPOINT;
    cmd->cmd.instrumentation.cmd.setpoint.channel = ch;
    cmd->cmd.instrumentation.cmd.setpoint.val = val;
    DEBUG_PRINTF(("<main.c> read_rts_command INSTRUMENTATION_COMMAND SETPOINT div=%u channel
        ↪ =%u val=%u\n",
                  cmd->instrumentation_division,
                  cmd->cmd.instrumentation.cmd.setpoint.channel,
                  cmd->cmd.instrumentation.cmd.setpoint.val));
    ok = 1;
#ifndef SIMULATE_SENSORS
  } else if (3 == (ok = sscanf(line, "V %hhd %hhd %d", &sensor, &ch, &val))) {
    if (sensor < 2 && ch < 2)
      sensors[ch][sensor] = val;
    DEBUG_PRINTF(("<main.c> read_rts_command UPDATE SENSORS sensor=%d, ch=%d, val=%d\n",
```

```
150 |         sensor,ch,val));
151 | #endif
152 |   } else if (line[0] == 'Q') {
153 |     // printf("<main.c> read_rts_command QUIT\n");
154 |     exit(0);
155 |   } else if (line[0] == 'D') {
156 |     DEBUG_PRINTF(("<main.c> read_rts_command UPDATE DISPLAY\n"));
157 |     update_display();
158 |   } else if (3 == (ok = sscanf(line, "ES %hhd %hhd %hhd", &sensor, &ch, &mode))) {
159 |     error_sensor_mode[ch][sensor] = mode;
160 |     DEBUG_PRINTF(("<main.c> read_rts_command ERROR SENSOR sensor=%d, ch=%d, mode=%d\n",
161 |           sensor,ch,mode));
162 |   } else if (2 == (ok = sscanf(line, "EI %hhd %hhd", &div, &mode))) {
163 |     error_instrumentation_mode[div] = mode;
164 |     DEBUG_PRINTF(("<main.c> read_rts_command ERROR INSTRUMENTATION div=%d, mode=%d\n",
165 |           div,mode));
166 |   }
167 |
168 |   if (line)
169 |     free(line);
170 |
171 |   return ok;
172 | }
173 |
174 | void update_instrumentation_errors(void) {
175 |   for (int i = 0; i < NINSTR; ++i) {
176 |     if(error_instrumentation_mode[i] == 2) {
177 |       error_instrumentation[i] |= rand() % 2;
178 |     } else {
179 |       error_instrumentation[i] = error_instrumentation_mode[i];
180 |     }
181 |   }
182 | }
183 |
184 | void update_sensor_errors(void) {
185 |   for (int c = 0; c < 2; ++c) {
186 |     for (int s = 0; s < 2; ++s) {
187 |       switch (error_sensor_mode[c][s]) {
188 |         case 0:
189 |           error_sensor[c][s] = 0;
190 |           error_sensor_demux[c][s][0] = 0;
191 |           error_sensor_demux[c][s][1] = 0;
192 |           break;
193 |         case 1:
194 |           error_sensor[c][s] = 0;
195 |           error_sensor_demux[c][s][0] = 1;
196 |           error_sensor_demux[c][s][1] = 0;
197 |           break;
198 |         case 2:
199 |           error_sensor[c][s] = 0;
200 |           error_sensor_demux[c][s][0] = 0;
201 |           error_sensor_demux[c][s][1] = 1;
202 |           break;
203 |         case 3:
204 |           error_sensor[c][s] = 1;
205 |           error_sensor_demux[c][s][0] = 0;
206 |           error_sensor_demux[c][s][1] = 0;
207 |           break;
208 |         case 4:
209 |         {
210 |           int fail = rand() % 2;
211 |           error_sensor[c][s] |= fail;
212 |           error_sensor_demux[c][s][0] = 0;
213 |           error_sensor_demux[c][s][1] = 0;
214 |         }
215 |         break;
```

```
      case 5:
      {
        error_sensor[c][s] = 0;
        error_sensor_demux[c][s][0] |= (rand() % 2);
        error_sensor_demux[c][s][1] |= (rand() % 2);
      }
      break;
      default:
        ASSERT("Invalid sensor fail mode" && 0);
    }
  }
 }
}

int update_sensor_simulation(void) {
  static int initialized = 0;
  static uint32_t last_update = 0;
  static uint32_t last[2][2] = {0};

  struct timespec tp;
  clock_gettime(CLOCK_REALTIME, &tp);
  uint32_t t0 = last_update;
  uint32_t t = tp.tv_sec*1000 + tp.tv_nsec/1000000;

  if (!initialized) {
    last_update = t;
    last[0][T] = T0;
    last[1][T] = T0;
    last[0][P] = P0;
    last[1][P] = P0;
    initialized = 1;
  } else if (t - t0 > SENSOR_UPDATE_MS) {
    for (int s = 0; s < 2; ++s) {
      last[s][T] += (rand() % 7) - 3 + T_BIAS;
      // Don't stray too far from our steam table
      last[s][T] = min(last[s][T], 300);
      last[s][T] = max(last[s][T], 25);

      last[s][P] += (rand() % 7) - 3 + P_BIAS;
      // Don't stray too far from our steam table
      last[s][P] = min(last[s][P], 5775200);
      last[s][P] = max(last[s][P], 8000);
    }
    last_update = t;
  }
  sensors[T][0] = last[T][0];
  sensors[T][1] = last[T][1];
  sensors[P][0] = last[P][0];
  sensors[P][1] = last[P][1];

  return 0;
}

void update_sensors(void) {
  update_sensor_errors();
#ifdef SIMULATE_SENSORS
  update_sensor_simulation();
#endif
  for (int c = 0; c < 2; ++c) {
    for (int s = 0; s < 2; ++s) {
      if (error_sensor[c][s]) {
        sensors[c][s] = rand();
      }

      MUTEX_LOCK(&mem_mutex);
      sensors_demux[c][s][0] = sensors[c][s];
```

```
      MUTEX_UNLOCK(&mem_mutex);

      MUTEX_LOCK(&mem_mutex);
      sensors_demux[c][s][1] = sensors[c][s];
      MUTEX_UNLOCK(&mem_mutex);

      for (int d = 0; d < 2; ++d) {
        if(error_sensor_demux[c][s][d]) {
          MUTEX_LOCK(&mem_mutex);
          sensors_demux[c][s][d] = rand();
          MUTEX_UNLOCK(&mem_mutex);
        }
      }
    }
  }
}

int read_actuation_command(uint8_t id, struct actuation_command *cmd) {
  struct actuation_command *c = act_command_buf[id];
  if (c) {
    cmd->device = c->device;
    cmd->on = c->on;
    free(c);
    act_command_buf[id] = NULL;
    return 1;
  }
  return 0;
}

int send_actuation_command(uint8_t id, struct actuation_command *cmd) {
  if (id < 2) {
    act_command_buf[id] = (struct actuation_command *)malloc(sizeof(*act_command_buf[id]));
    act_command_buf[id]->device = cmd->device;
    act_command_buf[id]->on = cmd->on;
    return 0;
  }
  return -1;
}

void* start0(void *arg) {
  while(1) {
    sense_actuate_step_0(&instrumentation[0], &actuation_logic[0]);
  }
}
void* start1(void *arg) {
  while(1) {
    sense_actuate_step_1(&instrumentation[2], &actuation_logic[1]);
  }
}

uint32_t time_in_s()
{
  static time_t start_time = 0;
  struct timespec tp;
  clock_gettime(CLOCK_REALTIME, &tp);
  if (start_time == 0) {
    start_time = tp.tv_sec;
  }
  time_t total = tp.tv_sec - start_time;
  char line[256];
  sprintf(line, "Uptime: [%u]s\n",(uint32_t)total);
  set_display_line(&core.ui, 9, line, 0);
  return (uint32_t)total;
}

int main(int argc, char **argv) {
```

```
  struct rts_command *cmd = (struct rts_command *)malloc(sizeof(*cmd));

  core_init(&core);
  sense_actuate_init(0, &instrumentation[0], &actuation_logic[0]);
  sense_actuate_init(1, &instrumentation[2], &actuation_logic[1]);

  if (isatty(fileno(stdin))) printf("\e[1;1H\e[2J");
  if (isatty(fileno(stdin))) printf("\e[%d;3H\e[2K> ", NLINES+1);

#ifdef USE_PTHREADS
  pthread_attr_t attr;
  pthread_t sense_actuate_0, sense_actuate_1;
  pthread_attr_init(&attr);
  pthread_create(&sense_actuate_0, &attr, start0, NULL);
  pthread_create(&sense_actuate_1, &attr, start1, NULL);
#endif

  while (1) {
    char line[256];
    fflush(stdout);
    MUTEX_LOCK(&display_mutex);
    sprintf(line, "HW ACTUATORS %s %s", actuator_state[0] ? "ON " : "OFF", actuator_state[1]?
        ↪  "ON " : "OFF");
    set_display_line(&core.ui, 8, line, 0);
    MUTEX_UNLOCK(&display_mutex);
    update_instrumentation_errors();
    update_sensors();
    core_step(&core);
#ifndef USE_PTHREADS
    sense_actuate_step_0(&instrumentation[0], &actuation_logic[0]);
    sense_actuate_step_1(&instrumentation[2], &actuation_logic[1]);
#endif
    update_display();
    sleep(1);
  }

  return 0;
}
```

Listing F.4: C implementation of instrumentation_common.

```
#include "instrumentation.h"

void instrumentation_init(struct instrumentation_state *state) {
  state->maintenance = 1;
  for (int i = 0; i < NTRIP; ++i) {
    state->mode[i] = 0;
    state->reading[i] = 0;
    state->sensor_trip[i] = 0;
    state->setpoints[i] = 0;
  }
}
```

Listing F.5: C implementation of syscalls.

```
#include <errno.h>
#include <sys/stat.h>
#include <sys/times.h>
#include <sys/time.h>
#include "syscalls.h"
#include "bsp.h"

void _exit(int n) {
    (void)n;
    while(1){
```

```
        ;;
    }
}

void *_sbrk(int nbytes)
{
    (void)nbytes;
    errno = ENOMEM;
    return (void *)-1;
}

int _write(int file, char *ptr, int len)
{
    volatile uint32_t *uart_tx = (void*) UART_REG_TX;
    (void)file;
    for (int i=0;i<len;i++) {
        *uart_tx = ptr[i];
        uint32_t count = MIN_PRINT_DELAY_TICKS;
        while(count-->0) {
            __asm__ volatile ("nop");
        }
        }
    return len;
}

int _close(int fd)
{
    (void)fd;
    errno = EBADF;
    return -1;
}

long _lseek(int fd, long offset, int origin)
{
    (void)fd;
    (void)offset;
    (void)origin;
    errno = EBADF;
    return -1;
}

int _read(int fd, void *buffer, unsigned int count)
{
    (void)fd;
    (void)buffer;
    (void)count;
    errno = EBADF;
    return -1;
}

int _fstat(int fd, void *buffer)
{
    (void)fd;
    (void)buffer;
    errno = EBADF;
    return -1;
}

int _isatty(int fd)
{
    (void)fd;
    errno = EBADF;
    return 0;
}

int _kill(int pid, int sig)
```

```
{
    (void)pid;
    (void)sig;
    errno = EBADF;
    return -1;
}

int _getpid(int n)
{
    (void)n;
    return 1;
}
```

Listing F.6: C implementation of printf.

```
// File from: https://github.com/mpaland/printf
//////////////////////////////////////////////////////////////////////////////
// \author (c) Marco Paland (info@paland.com)
// 2014-2019, PALANDesign Hannover, Germany
//
// \license The MIT License (MIT)
//
// Permission is hereby granted, free of charge, to any person obtaining a copy
// of this software and associated documentation files (the "Software"), to deal
// in the Software without restriction, including without limitation the rights
// to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
// copies of the Software, and to permit persons to whom the Software is
// furnished to do so, subject to the following conditions:
//
// The above copyright notice and this permission notice shall be included in
// all copies or substantial portions of the Software.
//
// THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
// IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
// FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
// AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
// LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
// OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN
// THE SOFTWARE.
//
// \brief Tiny printf, sprintf and (v)snprintf implementation, optimized for speed on
// embedded systems with a very limited resources. These routines are thread
// safe and reentrant!
// Use this instead of the bloated standard/newlib printf cause these use
// malloc for printf (and may not be thread safe).
//
//////////////////////////////////////////////////////////////////////////////

#include <stdbool.h>
#include <stdint.h>

#include "printf.h"
#include "syscalls.h"

// define this globally (e.g. gcc -DPRINTF_INCLUDE_CONFIG_H ...) to include the
// printf_config.h header file
// default: undefined
#ifdef PRINTF_INCLUDE_CONFIG_H
#include "printf_config.h"
#endif

// 'ntoa' conversion buffer size, this must be big enough to hold one converted
// numeric number including padded zeros (dynamically created on stack)
// default: 32 byte
#ifndef PRINTF_NTOA_BUFFER_SIZE
#define PRINTF_NTOA_BUFFER_SIZE 32U
```

```
#endif

// 'ftoa' conversion buffer size, this must be big enough to hold one converted
// float number including padded zeros (dynamically created on stack)
// default: 32 byte
#ifndef PRINTF_FTOA_BUFFER_SIZE
#define PRINTF_FTOA_BUFFER_SIZE 32U
#endif

// support for the floating point type (%f)
// default: activated
#ifndef PRINTF_DISABLE_SUPPORT_FLOAT
#define PRINTF_SUPPORT_FLOAT
#endif

// support for exponential floating point notation (%e/%g)
// default: activated
#ifndef PRINTF_DISABLE_SUPPORT_EXPONENTIAL
#define PRINTF_SUPPORT_EXPONENTIAL
#endif

// define the default floating point precision
// default: 6 digits
#ifndef PRINTF_DEFAULT_FLOAT_PRECISION
#define PRINTF_DEFAULT_FLOAT_PRECISION 6U
#endif

// define the largest float suitable to print with %f
// default: 1e9
#ifndef PRINTF_MAX_FLOAT
#define PRINTF_MAX_FLOAT 1e9
#endif

// support for the long long types (%llu or %p)
// default: activated
#ifndef PRINTF_DISABLE_SUPPORT_LONG_LONG
#define PRINTF_SUPPORT_LONG_LONG
#endif

// support for the ptrdiff_t type (%t)
// ptrdiff_t is normally defined in <stddef.h> as long or long long type
// default: activated
#ifndef PRINTF_DISABLE_SUPPORT_PTRDIFF_T
#define PRINTF_SUPPORT_PTRDIFF_T
#endif

///////////////////////////////////////////////////////////////////////////////

// internal flag definitions
#define FLAGS_ZEROPAD (1U << 0U)
#define FLAGS_LEFT (1U << 1U)
#define FLAGS_PLUS (1U << 2U)
#define FLAGS_SPACE (1U << 3U)
#define FLAGS_HASH (1U << 4U)
#define FLAGS_UPPERCASE (1U << 5U)
#define FLAGS_CHAR (1U << 6U)
#define FLAGS_SHORT (1U << 7U)
#define FLAGS_LONG (1U << 8U)
#define FLAGS_LONG_LONG (1U << 9U)
#define FLAGS_PRECISION (1U << 10U)
#define FLAGS_ADAPT_EXP (1U << 11U)

// import float.h for DBL_MAX
#if defined(PRINTF_SUPPORT_FLOAT)
#include <float.h>
#endif
```

```
118 |
119 | // output function type
120 | typedef void (*out_fct_type)(char character, void* buffer, size_t idx, size_t maxlen);
121 |
122 | // wrapper (used as buffer) for output function type
123 | typedef struct {
124 |   void (*fct)(char character, void* arg);
125 |   void* arg;
126 | } out_fct_wrap_type;
127 |
128 | // internal buffer output
129 | static inline void _out_buffer(char character, void* buffer, size_t idx, size_t maxlen)
130 | {
131 |   if (idx < maxlen) {
132 |     ((char*)buffer)[idx] = character;
133 |   }
134 | }
135 |
136 | // internal null output
137 | static inline void _out_null(char character, void* buffer, size_t idx, size_t maxlen)
138 | {
139 |   (void)character; (void)buffer; (void)idx; (void)maxlen;
140 | }
141 |
142 | // internal _putchar wrapper
143 | static inline void _out_char(char character, void* buffer, size_t idx, size_t maxlen)
144 | {
145 |   (void)buffer; (void)idx; (void)maxlen;
146 |   if (character) {
147 |     _write(0, &character, 1);
148 |   }
149 | }
150 |
151 | // internal output function wrapper
152 | static inline void _out_fct(char character, void* buffer, size_t idx, size_t maxlen)
153 | {
154 |   (void)idx; (void)maxlen;
155 |   if (character) {
156 |     // buffer is the output fct pointer
157 |     ((out_fct_wrap_type*)buffer)->fct(character, ((out_fct_wrap_type*)buffer)->arg);
158 |   }
159 | }
160 |
161 | // internal secure strlen
162 | // \return The length of the string (excluding the terminating 0) limited by 'maxsize'
163 | static inline unsigned int _strnlen_s(const char* str, size_t maxsize)
164 | {
165 |   const char* s;
166 |   for (s = str; *s && maxsize--; ++s);
167 |   return (unsigned int)(s - str);
168 | }
169 |
170 | // internal test if char is a digit (0-9)
171 | // \return true if char is a digit
172 | static inline bool _is_digit(char ch)
173 | {
174 |   return (ch >= '0') && (ch <= '9');
175 | }
176 |
177 | // internal ASCII string to unsigned int conversion
178 | static unsigned int _atoi(const char** str)
179 | {
180 |   unsigned int i = 0U;
181 |   while (_is_digit(**str)) {
182 |     i = i * 10U + (unsigned int)(*((*str)++) - '0');
183 |   }
```

```
184   return i;
185 }
186
187 // output the specified string in reverse, taking care of any zero-padding
188 static size_t _out_rev(out_fct_type out, char* buffer, size_t idx, size_t maxlen, const char
        ↪ * buf, size_t len, unsigned int width, unsigned int flags)
189 {
190   const size_t start_idx = idx;
191
192   // pad spaces up to given width
193   if (!(flags & FLAGS_LEFT) && !(flags & FLAGS_ZEROPAD)) {
194     for (size_t i = len; i < width; i++) {
195       out(' ', buffer, idx++, maxlen);
196     }
197   }
198
199   // reverse string
200   while (len) {
201     out(buf[--len], buffer, idx++, maxlen);
202   }
203
204   // append pad spaces up to given width
205   if (flags & FLAGS_LEFT) {
206     while (idx - start_idx < width) {
207       out(' ', buffer, idx++, maxlen);
208     }
209   }
210
211   return idx;
212 }
213
214 // internal itoa format
215 static size_t _ntoa_format(out_fct_type out, char* buffer, size_t idx, size_t maxlen, char*
        ↪ buf, size_t len, bool negative, unsigned int base, unsigned int prec, unsigned int
        ↪ width, unsigned int flags)
216 {
217   // pad leading zeros
218   if (!(flags & FLAGS_LEFT)) {
219     if (width && (flags & FLAGS_ZEROPAD) && (negative || (flags & (FLAGS_PLUS | FLAGS_SPACE))
            ↪ )) {
220       width--;
221     }
222     while ((len < prec) && (len < PRINTF_NTOA_BUFFER_SIZE)) {
223       buf[len++] = '0';
224     }
225     while ((flags & FLAGS_ZEROPAD) && (len < width) && (len < PRINTF_NTOA_BUFFER_SIZE)) {
226       buf[len++] = '0';
227     }
228   }
229
230   // handle hash
231   if (flags & FLAGS_HASH) {
232     if (!(flags & FLAGS_PRECISION) && len && ((len == prec) || (len == width))) {
233       len--;
234       if (len && (base == 16U)) {
235         len--;
236       }
237     }
238     if ((base == 16U) && !(flags & FLAGS_UPPERCASE) && (len < PRINTF_NTOA_BUFFER_SIZE)) {
239       buf[len++] = 'x';
240     }
241     else if ((base == 16U) && (flags & FLAGS_UPPERCASE) && (len < PRINTF_NTOA_BUFFER_SIZE)) {
242       buf[len++] = 'X';
243     }
244     else if ((base == 2U) && (len < PRINTF_NTOA_BUFFER_SIZE)) {
245       buf[len++] = 'b';
```

```
246     }
247     if (len < PRINTF_NTOA_BUFFER_SIZE) {
248       buf[len++] = '0';
249     }
250   }
251
252   if (len < PRINTF_NTOA_BUFFER_SIZE) {
253     if (negative) {
254       buf[len++] = '-';
255     }
256     else if (flags & FLAGS_PLUS) {
257       buf[len++] = '+'; // ignore the space if the '+' exists
258     }
259     else if (flags & FLAGS_SPACE) {
260       buf[len++] = ' ';
261     }
262   }
263
264   return _out_rev(out, buffer, idx, maxlen, buf, len, width, flags);
265 }
266
267 // internal itoa for 'long' type
268 static size_t _ntoa_long(out_fct_type out, char* buffer, size_t idx, size_t maxlen, unsigned
      ↪  long value, bool negative, unsigned long base, unsigned int prec, unsigned int
      ↪ width, unsigned int flags)
269 {
270   char buf[PRINTF_NTOA_BUFFER_SIZE];
271   size_t len = 0U;
272
273   // no hash for 0 values
274   if (!value) {
275     flags &= ~FLAGS_HASH;
276   }
277
278   // write if precision != 0 and value is != 0
279   if (!(flags & FLAGS_PRECISION) || value) {
280     do {
281       const char digit = (char)(value % base);
282       buf[len++] = digit < 10 ? '0' + digit : (flags & FLAGS_UPPERCASE ? 'A' : 'a') + digit -
            ↪  10;
283       value /= base;
284     } while (value && (len < PRINTF_NTOA_BUFFER_SIZE));
285   }
286
287   return _ntoa_format(out, buffer, idx, maxlen, buf, len, negative, (unsigned int)base, prec
        ↪ , width, flags);
288 }
289
290 // internal itoa for 'long long' type
291 #if defined(PRINTF_SUPPORT_LONG_LONG)
292 static size_t _ntoa_long_long(out_fct_type out, char* buffer, size_t idx, size_t maxlen,
      ↪ unsigned long long value, bool negative, unsigned long long base, unsigned int prec,
      ↪  unsigned int width, unsigned int flags)
293 {
294   char buf[PRINTF_NTOA_BUFFER_SIZE];
295   size_t len = 0U;
296
297   // no hash for 0 values
298   if (!value) {
299     flags &= ~FLAGS_HASH;
300   }
301
302   // write if precision != 0 and value is != 0
303   if (!(flags & FLAGS_PRECISION) || value) {
304     do {
305       const char digit = (char)(value % base);
```

```
306       buf[len++] = digit < 10 ? '0' + digit : (flags & FLAGS_UPPERCASE ? 'A' : 'a') + digit -
              ↪ 10;
307       value /= base;
308     } while (value && (len < PRINTF_NTOA_BUFFER_SIZE));
309   }
310
311   return _ntoa_format(out, buffer, idx, maxlen, buf, len, negative, (unsigned int)base, prec
          ↪ , width, flags);
312 }
313 #endif // PRINTF_SUPPORT_LONG_LONG
314
315 #if defined(PRINTF_SUPPORT_FLOAT)
316
317 #if defined(PRINTF_SUPPORT_EXPONENTIAL)
318 // forward declaration so that _ftoa can switch to exp notation for values > PRINTF_MAX_FLOAT
319 static size_t _etoa(out_fct_type out, char* buffer, size_t idx, size_t maxlen, double value,
          ↪ unsigned int prec, unsigned int width, unsigned int flags);
320 #endif
321
322 // internal ftoa for fixed decimal floating point
323 static size_t _ftoa(out_fct_type out, char* buffer, size_t idx, size_t maxlen, double value,
          ↪ unsigned int prec, unsigned int width, unsigned int flags)
324 {
325   char buf[PRINTF_FTOA_BUFFER_SIZE];
326   size_t len = 0U;
327   double diff = 0.0;
328
329   // powers of 10
330   static const double pow10[] = { 1, 10, 100, 1000, 10000, 100000, 1000000, 10000000,
          ↪ 100000000, 1000000000 };
331
332   // test for special values
333   if (value != value)
334     return _out_rev(out, buffer, idx, maxlen, "nan", 3, width, flags);
335   if (value < -DBL_MAX)
336     return _out_rev(out, buffer, idx, maxlen, "fni-", 4, width, flags);
337   if (value > DBL_MAX)
338     return _out_rev(out, buffer, idx, maxlen, (flags & FLAGS_PLUS) ? "fni+" : "fni", (flags &
          ↪ FLAGS_PLUS) ? 4U : 3U, width, flags);
339
340   // test for very large values
341   // standard printf behavior is to print EVERY whole number digit -- which could be 100s of
          ↪ characters overflowing your buffers == bad
342   if ((value > PRINTF_MAX_FLOAT) || (value < -PRINTF_MAX_FLOAT)) {
343 #if defined(PRINTF_SUPPORT_EXPONENTIAL)
344     return _etoa(out, buffer, idx, maxlen, value, prec, width, flags);
345 #else
346     return 0U;
347 #endif
348   }
349
350   // test for negative
351   bool negative = false;
352   if (value < 0) {
353     negative = true;
354     value = 0 - value;
355   }
356
357   // set default precision, if not set explicitly
358   if (!(flags & FLAGS_PRECISION)) {
359     prec = PRINTF_DEFAULT_FLOAT_PRECISION;
360   }
361   // limit precision to 9, cause a prec >= 10 can lead to overflow errors
362   while ((len < PRINTF_FTOA_BUFFER_SIZE) && (prec > 9U)) {
363     buf[len++] = '0';
364     prec--;
365   }
```

```
366 |
367 |   int whole = (int)value;
368 |   double tmp = (value - whole) * pow10[prec];
369 |   unsigned long frac = (unsigned long)tmp;
370 |   diff = tmp - frac;
371 |
372 |   if (diff > 0.5) {
373 |     ++frac;
374 |     // handle rollover, e.g. case 0.99 with prec 1 is 1.0
375 |     if (frac >= pow10[prec]) {
376 |       frac = 0;
377 |       ++whole;
378 |     }
379 |   }
380 |   else if (diff < 0.5) {
381 |   }
382 |   else if ((frac == 0U) || (frac & 1U)) {
383 |     // if halfway, round up if odd OR if last digit is 0
384 |     ++frac;
385 |   }
386 |
387 |   if (prec == 0U) {
388 |     diff = value - (double)whole;
389 |     if ((!(diff < 0.5) || (diff > 0.5)) && (whole & 1)) {
390 |       // exactly 0.5 and ODD, then round up
391 |       // 1.5 -> 2, but 2.5 -> 2
392 |       ++whole;
393 |     }
394 |   }
395 |   else {
396 |     unsigned int count = prec;
397 |     // now do fractional part, as an unsigned number
398 |     while (len < PRINTF_FTOA_BUFFER_SIZE) {
399 |       --count;
400 |       buf[len++] = (char)(48U + (frac % 10U));
401 |       if (!(frac /= 10U)) {
402 |         break;
403 |       }
404 |     }
405 |     // add extra 0s
406 |     while ((len < PRINTF_FTOA_BUFFER_SIZE) && (count-- > 0U)) {
407 |       buf[len++] = '0';
408 |     }
409 |     if (len < PRINTF_FTOA_BUFFER_SIZE) {
410 |       // add decimal
411 |       buf[len++] = '.';
412 |     }
413 |   }
414 |
415 |   // do whole part, number is reversed
416 |   while (len < PRINTF_FTOA_BUFFER_SIZE) {
417 |     buf[len++] = (char)(48 + (whole % 10));
418 |     if (!(whole /= 10)) {
419 |       break;
420 |     }
421 |   }
422 |
423 |   // pad leading zeros
424 |   if (!(flags & FLAGS_LEFT) && (flags & FLAGS_ZEROPAD)) {
425 |     if (width && (negative || (flags & (FLAGS_PLUS | FLAGS_SPACE)))) {
426 |       width--;
427 |     }
428 |     while ((len < width) && (len < PRINTF_FTOA_BUFFER_SIZE)) {
429 |       buf[len++] = '0';
430 |     }
431 |   }
```

```
  if (len < PRINTF_FTOA_BUFFER_SIZE) {
    if (negative) {
      buf[len++] = '-';
    }
    else if (flags & FLAGS_PLUS) {
      buf[len++] = '+'; // ignore the space if the '+' exists
    }
    else if (flags & FLAGS_SPACE) {
      buf[len++] = ' ';
    }
  }

  return _out_rev(out, buffer, idx, maxlen, buf, len, width, flags);
}

#if defined(PRINTF_SUPPORT_EXPONENTIAL)
// internal ftoa variant for exponential floating-point type, contributed by Martijn Jasperse <m.jasperse@gmail.com>
static size_t _etoa(out_fct_type out, char* buffer, size_t idx, size_t maxlen, double value, unsigned int prec, unsigned int width, unsigned int flags)
{
  // check for NaN and special values
  if ((value != value) || (value > DBL_MAX) || (value < -DBL_MAX)) {
    return _ftoa(out, buffer, idx, maxlen, value, prec, width, flags);
  }

  // determine the sign
  const bool negative = value < 0;
  if (negative) {
    value = -value;
  }

  // default precision
  if (!(flags & FLAGS_PRECISION)) {
    prec = PRINTF_DEFAULT_FLOAT_PRECISION;
  }

  // determine the decimal exponent
  // based on the algorithm by David Gay (https://www.ampl.com/netlib/fp/dtoa.c)
  union {
    uint64_t U;
    double F;
  } conv;

  conv.F = value;
  int exp2 = (int)((conv.U >> 52U) & 0x07FFU) - 1023; // effectively log2
  conv.U = (conv.U & ((1ULL << 52U) - 1U)) | (1023ULL << 52U); // drop the exponent so conv.F is now in [1,2)
  // now approximate log10 from the log2 integer part and an expansion of ln around 1.5
  int expval = (int)(0.1760912590558 + exp2 * 0.301029995663981 + (conv.F - 1.5) * 0.289529654602168);
  // now we want to compute 10^expval but we want to be sure it won't overflow
  exp2 = (int)(expval * 3.321928094887362 + 0.5);
  const double z = expval * 2.302585092994046 - exp2 * 0.6931471805599453;
  const double z2 = z * z;
  conv.U = (uint64_t)(exp2 + 1023) << 52U;
  // compute exp(z) using continued fractions, see https://en.wikipedia.org/wiki/Exponential_function#Continued_fractions_for_ex
  conv.F *= 1 + 2 * z / (2 - z + (z2 / (6 + (z2 / (10 + z2 / 14)))));
  // correct for rounding errors
  if (value < conv.F) {
    expval--;
    conv.F /= 10;
  }

```

```
493   // the exponent format is "%+03d" and largest value is "307", so set aside 4-5 characters
494   unsigned int minwidth = ((expval < 100) && (expval > -100)) ? 4U : 5U;
495
496   // in "%g" mode, "prec" is the number of *significant figures* not decimals
497   if (flags & FLAGS_ADAPT_EXP) {
498     // do we want to fall-back to "%f" mode?
499     if ((value >= 1e-4) && (value < 1e6)) {
500       if ((int)prec > expval) {
501         prec = (unsigned)((int)prec - expval - 1);
502       }
503       else {
504         prec = 0;
505       }
506       flags |= FLAGS_PRECISION; // make sure _ftoa respects precision
507       // no characters in exponent
508       minwidth = 0U;
509       expval = 0;
510     }
511     else {
512       // we use one sigfig for the whole part
513       if ((prec > 0) && (flags & FLAGS_PRECISION)) {
514         --prec;
515       }
516     }
517   }
518
519   // will everything fit?
520   unsigned int fwidth = width;
521   if (width > minwidth) {
522     // we didn't fall-back so subtract the characters required for the exponent
523     fwidth -= minwidth;
524   } else {
525     // not enough characters, so go back to default sizing
526     fwidth = 0U;
527   }
528   if ((flags & FLAGS_LEFT) && minwidth) {
529     // if we're padding on the right, DON'T pad the floating part
530     fwidth = 0U;
531   }
532
533   // rescale the float value
534   if (expval) {
535     value /= conv.F;
536   }
537
538   // output the floating part
539   const size_t start_idx = idx;
540   idx = _ftoa(out, buffer, idx, maxlen, negative ? -value : value, prec, fwidth, flags & ~
          ↪ FLAGS_ADAPT_EXP);
541
542   // output the exponent part
543   if (minwidth) {
544     // output the exponential symbol
545     out((flags & FLAGS_UPPERCASE) ? 'E' : 'e', buffer, idx++, maxlen);
546     // output the exponent value
547     idx = _ntoa_long(out, buffer, idx, maxlen, (expval < 0) ? -expval : expval, expval < 0,
            ↪ 10, 0, minwidth-1, FLAGS_ZEROPAD | FLAGS_PLUS);
548     // might need to right-pad spaces
549     if (flags & FLAGS_LEFT) {
550       while (idx - start_idx < width) out(' ', buffer, idx++, maxlen);
551     }
552   }
553   return idx;
554 }
555 #endif // PRINTF_SUPPORT_EXPONENTIAL
556 #endif // PRINTF_SUPPORT_FLOAT
```

```
557 |
558 | // internal vsnprintf
559 | static int _vsnprintf(out_fct_type out, char* buffer, const size_t maxlen, const char*
      ↪ format, va_list va)
560 | {
561 |   unsigned int flags, width, precision, n;
562 |   size_t idx = 0U;
563 |
564 |   if (!buffer) {
565 |     // use null output function
566 |     out = _out_null;
567 |   }
568 |
569 |   while (*format)
570 |   {
571 |     // format specifier? %[flags][width][.precision][length]
572 |     if (*format != '%') {
573 |       // no
574 |       out(*format, buffer, idx++, maxlen);
575 |       format++;
576 |       continue;
577 |     }
578 |     else {
579 |       // yes, evaluate it
580 |       format++;
581 |     }
582 |
583 |     // evaluate flags
584 |     flags = 0U;
585 |     do {
586 |       switch (*format) {
587 |         case '0': flags |= FLAGS_ZEROPAD; format++; n = 1U; break;
588 |         case '-': flags |= FLAGS_LEFT; format++; n = 1U; break;
589 |         case '+': flags |= FLAGS_PLUS; format++; n = 1U; break;
590 |         case ' ': flags |= FLAGS_SPACE; format++; n = 1U; break;
591 |         case '#': flags |= FLAGS_HASH; format++; n = 1U; break;
592 |         default : n = 0U; break;
593 |       }
594 |     } while (n);
595 |
596 |     // evaluate width field
597 |     width = 0U;
598 |     if (_is_digit(*format)) {
599 |       width = _atoi(&format);
600 |     }
601 |     else if (*format == '*') {
602 |       const int w = va_arg(va, int);
603 |       if (w < 0) {
604 |         flags |= FLAGS_LEFT; // reverse padding
605 |         width = (unsigned int)-w;
606 |       }
607 |       else {
608 |         width = (unsigned int)w;
609 |       }
610 |       format++;
611 |     }
612 |
613 |     // evaluate precision field
614 |     precision = 0U;
615 |     if (*format == '.') {
616 |       flags |= FLAGS_PRECISION;
617 |       format++;
618 |       if (_is_digit(*format)) {
619 |         precision = _atoi(&format);
620 |       }
621 |       else if (*format == '*') {
```

```
622         const int prec = (int)va_arg(va, int);
623         precision = prec > 0 ? (unsigned int)prec : 0U;
624         format++;
625       }
626     }
627 
628     // evaluate length field
629     switch (*format) {
630       case 'l' :
631         flags |= FLAGS_LONG;
632         format++;
633         if (*format == 'l') {
634           flags |= FLAGS_LONG_LONG;
635           format++;
636         }
637         break;
638       case 'h' :
639         flags |= FLAGS_SHORT;
640         format++;
641         if (*format == 'h') {
642           flags |= FLAGS_CHAR;
643           format++;
644         }
645         break;
646 #if defined(PRINTF_SUPPORT_PTRDIFF_T)
647       case 't' :
648         flags |= (sizeof(ptrdiff_t) == sizeof(long) ? FLAGS_LONG : FLAGS_LONG_LONG);
649         format++;
650         break;
651 #endif
652       case 'j' :
653         flags |= (sizeof(intmax_t) == sizeof(long) ? FLAGS_LONG : FLAGS_LONG_LONG);
654         format++;
655         break;
656       case 'z' :
657         flags |= (sizeof(size_t) == sizeof(long) ? FLAGS_LONG : FLAGS_LONG_LONG);
658         format++;
659         break;
660       default :
661         break;
662     }
663 
664     // evaluate specifier
665     switch (*format) {
666       case 'd' :
667       case 'i' :
668       case 'u' :
669       case 'x' :
670       case 'X' :
671       case 'o' :
672       case 'b' : {
673         // set the base
674         unsigned int base;
675         if (*format == 'x' || *format == 'X') {
676           base = 16U;
677         }
678         else if (*format == 'o') {
679           base = 8U;
680         }
681         else if (*format == 'b') {
682           base = 2U;
683         }
684         else {
685           base = 10U;
686           flags &= ~FLAGS_HASH; // no hash for dec format
687         }
```

```
688         // uppercase
689         if (*format == 'X') {
690           flags |= FLAGS_UPPERCASE;
691         }
692
693         // no plus or space flag for u, x, X, o, b
694         if ((*format != 'i') && (*format != 'd')) {
695           flags &= ~(FLAGS_PLUS | FLAGS_SPACE);
696         }
697
698         // ignore '0' flag when precision is given
699         if (flags & FLAGS_PRECISION) {
700           flags &= ~FLAGS_ZEROPAD;
701         }
702
703         // convert the integer
704         if ((*format == 'i') || (*format == 'd')) {
705           // signed
706           if (flags & FLAGS_LONG_LONG) {
707 #if defined(PRINTF_SUPPORT_LONG_LONG)
708             const long long value = va_arg(va, long long);
709             idx = _ntoa_long_long(out, buffer, idx, maxlen, (unsigned long long)(value > 0 ?
                    ↪ value : 0 - value), value < 0, base, precision, width, flags);
710 #endif
711           }
712           else if (flags & FLAGS_LONG) {
713             const long value = va_arg(va, long);
714             idx = _ntoa_long(out, buffer, idx, maxlen, (unsigned long)(value > 0 ? value : 0
                    ↪ - value), value < 0, base, precision, width, flags);
715           }
716           else {
717             const int value = (flags & FLAGS_CHAR) ? (char)va_arg(va, int) : (flags &
                    ↪ FLAGS_SHORT) ? (short int)va_arg(va, int) : va_arg(va, int);
718             idx = _ntoa_long(out, buffer, idx, maxlen, (unsigned int)(value > 0 ? value : 0 -
                    ↪  value), value < 0, base, precision, width, flags);
719           }
720         }
721         else {
722           // unsigned
723           if (flags & FLAGS_LONG_LONG) {
724 #if defined(PRINTF_SUPPORT_LONG_LONG)
725             idx = _ntoa_long_long(out, buffer, idx, maxlen, va_arg(va, unsigned long long),
                    ↪ false, base, precision, width, flags);
726 #endif
727           }
728           else if (flags & FLAGS_LONG) {
729             idx = _ntoa_long(out, buffer, idx, maxlen, va_arg(va, unsigned long), false, base
                    ↪ , precision, width, flags);
730           }
731           else {
732             const unsigned int value = (flags & FLAGS_CHAR) ? (unsigned char)va_arg(va,
                    ↪ unsigned int) : (flags & FLAGS_SHORT) ? (unsigned short int)va_arg(va,
                    ↪ unsigned int) : va_arg(va, unsigned int);
733             idx = _ntoa_long(out, buffer, idx, maxlen, value, false, base, precision, width,
                    ↪ flags);
734           }
735         }
736         format++;
737         break;
738       }
739 #if defined(PRINTF_SUPPORT_FLOAT)
740       case 'f' :
741       case 'F' :
742         if (*format == 'F') flags |= FLAGS_UPPERCASE;
743         idx = _ftoa(out, buffer, idx, maxlen, va_arg(va, double), precision, width, flags);
744         format++;
745         break;
```

```
#if defined(PRINTF_SUPPORT_EXPONENTIAL)
      case 'e':
      case 'E':
      case 'g':
      case 'G':
        if ((*format == 'g')||(*format == 'G')) flags |= FLAGS_ADAPT_EXP;
        if ((*format == 'E')||(*format == 'G')) flags |= FLAGS_UPPERCASE;
        idx = _etoa(out, buffer, idx, maxlen, va_arg(va, double), precision, width, flags);
        format++;
        break;
#endif // PRINTF_SUPPORT_EXPONENTIAL
#endif // PRINTF_SUPPORT_FLOAT
      case 'c' : {
        unsigned int l = 1U;
        // pre padding
        if (!(flags & FLAGS_LEFT)) {
          while (l++ < width) {
            out(' ', buffer, idx++, maxlen);
          }
        }
        // char output
        out((char)va_arg(va, int), buffer, idx++, maxlen);
        // post padding
        if (flags & FLAGS_LEFT) {
          while (l++ < width) {
            out(' ', buffer, idx++, maxlen);
          }
        }
        format++;
        break;
      }

      case 's' : {
        const char* p = va_arg(va, char*);
        unsigned int l = _strnlen_s(p, precision ? precision : (size_t)-1);
        // pre padding
        if (flags & FLAGS_PRECISION) {
          l = (l < precision ? l : precision);
        }
        if (!(flags & FLAGS_LEFT)) {
          while (l++ < width) {
            out(' ', buffer, idx++, maxlen);
          }
        }
        // string output
        while ((*p != 0) && (!(flags & FLAGS_PRECISION) || precision--)) {
          out(*(p++), buffer, idx++, maxlen);
        }
        // post padding
        if (flags & FLAGS_LEFT) {
          while (l++ < width) {
            out(' ', buffer, idx++, maxlen);
          }
        }
        format++;
        break;
      }

      case 'p' : {
        width = sizeof(void*) * 2U;
        flags |= FLAGS_ZEROPAD | FLAGS_UPPERCASE;
#if defined(PRINTF_SUPPORT_LONG_LONG)
        const bool is_ll = sizeof(uintptr_t) == sizeof(long long);
        if (is_ll) {
          idx = _ntoa_long_long(out, buffer, idx, maxlen, (uintptr_t)va_arg(va, void*), false
              ↪ , 16U, precision, width, flags);
```

```
811         }
812         else {
813 #endif
814           idx = _ntoa_long(out, buffer, idx, maxlen, (unsigned long)((uintptr_t)va_arg(va,
                  ↪ void*)), false, 16U, precision, width, flags);
815 #if defined(PRINTF_SUPPORT_LONG_LONG)
816         }
817 #endif
818         format++;
819         break;
820       }
821
822       case '%' :
823         out('%', buffer, idx++, maxlen);
824         format++;
825         break;
826
827       default :
828         out(*format, buffer, idx++, maxlen);
829         format++;
830         break;
831     }
832   }
833
834   // termination
835   out((char)0, buffer, idx < maxlen ? idx : maxlen - 1U, maxlen);
836
837   // return written chars without terminating \0
838   return (int)idx;
839 }
840
841 ///////////////////////////////////////////////////////////////////////////////
842
843 int printf_(const char* format, ...)
844 {
845   va_list va;
846   va_start(va, format);
847   char buffer[1];
848   const int ret = _vsnprintf(_out_char, buffer, (size_t)-1, format, va);
849   va_end(va);
850   return ret;
851 }
852
853 int sprintf_(char* buffer, const char* format, ...)
854 {
855   va_list va;
856   va_start(va, format);
857   const int ret = _vsnprintf(_out_buffer, buffer, (size_t)-1, format, va);
858   va_end(va);
859   return ret;
860 }
861
862 int snprintf_(char* buffer, size_t count, const char* format, ...)
863 {
864   va_list va;
865   va_start(va, format);
866   const int ret = _vsnprintf(_out_buffer, buffer, count, format, va);
867   va_end(va);
868   return ret;
869 }
870
871 int vprintf_(const char* format, va_list va)
872 {
873   char buffer[1];
874   return _vsnprintf(_out_char, buffer, (size_t)-1, format, va);
875 }
```

```
876
877 int vsnprintf_(char* buffer, size_t count, const char* format, va_list va)
878 {
879   return _vsnprintf(_out_buffer, buffer, count, format, va);
880 }
881
882 int fctprintf(void (*out)(char character, void* arg), void* arg, const char* format, ...)
883 {
884   va_list va;
885   va_start(va, format);
886   const out_fct_wrap_type out_fct_wrap = { out, arg };
887   const int ret = _vsnprintf(_out_fct, (char*)(uintptr_t)&out_fct_wrap, (size_t)-1, format,
          ↪ va);
888   va_end(va);
889   return ret;
890 }
```

Listing F.7: C implementation of actuation_unit_impl.

```
1 #include "common.h"
2 #include "actuation_logic.h"
3 #include "../generated/C/actuation_unit_impl.c"
```

Listing F.8: C implementation of C_Imported_Functions.

```
1  // Copyright (c) 2013-2019 Bluespec, Inc. All Rights Reserved
2  // https://github.com/bluespec/Piccolo/blob/master/src_Testbench/Top/C_Imported_Functions.c
3  // Modified by @podhrmic
4
5  // ================================================================
6  // These are functions imported into BSV during Bluesim or Verilog simulation.
7  // See C_Imports.bsv for the corresponding 'import BDPI' declarations.
8
9  // There are several independent groups of functions below; the
10 // groups are separated by heavy dividers ('// *******')
11
12 // Below, 'dummy' args are not used, and are present only to appease
13 // some Verilog simulators that are finicky about 0-arg functions.
14
15 // ================================================================
16 // Includes from C library
17
18 // General
19 #include <unistd.h>
20 #include <stdlib.h>
21 #include <stdio.h>
22 #include <stdint.h>
23 #include <stdbool.h>
24 #include <inttypes.h>
25 #include <string.h>
26 #include <errno.h>
27 #include <time.h>
28 #include <termios.h>
29 #include <unistd.h>
30 #include <sys/types.h>
31 #include <poll.h>
32 #include <sched.h>
33
34 // ================================================================
35 // Includes for this project
36
37 #include "C_Imported_Functions.h"
38
39 // ****************************************************************
40 // ****************************************************************
```

```
// ************************************************************

// Functions for console I/O

// ============================================================
// c_trygetchar()
// Returns next input character (ASCII code) from the console.
// Returns 0 if no input is available.

// NOTE: Not needed right now
void print_tty(char* name, FILE * f) {
    printf("%s (fileno %d): ", name, fileno(f));
    if (isatty(fileno(f))) printf("TTY %s\n", ttyname(fileno(f)));
    else printf("not a TTY\n");
}

// TODO: set to 1 to try avoid line buffering
#define INIT_TERMIOS 0
uint8_t c_trygetchar (uint8_t dummy)
{
    uint8_t ch;
    ssize_t n;
    struct pollfd x_pollfd;
    const int fd_stdin = 0;

#if INIT_TERMIOS
    static bool init = false;
    if (!init) {
        print_tty("stdin ", stdin);
        print_tty("stdout", stdout);
        print_tty("stderr", stderr);

        struct termios tconf;

        // get original cooked/canonical mode values
        tcgetattr(fd_stdin,&tconf);

        // set options for raw mode
        tconf.c_lflag &= ~(ECHO | ICANON); /* no echo or edit */
        tconf.c_cc[VMIN] = 0;
        tconf.c_cc[VTIME] = 0;

        // put unit into raw mode ...
        tcsetattr(fd_stdin,TCSANOW,&tconf);
        printf("Terminal set to ~(ECHO | ICANON) mode\n");
        init = true;
    }
#endif

    // ----------------
    // Poll for input
    x_pollfd.fd = fd_stdin;
    x_pollfd.events = POLLRDNORM;
    x_pollfd.revents = 0;
    poll (& x_pollfd, 1, 1);

    //printf ("INFO: c_trygetchar: Polling for input\n");
    if ((x_pollfd.revents & POLLRDNORM) == 0) {
        //printf ("INFO: No input\n");
        return 0;
    }

    // ----------------
    // Input is available

    n = read (fd_stdin, & ch, 1);
```

```
107     if (n == 1) {
108         //printf ("INFO: got %c\n",ch);
109     return ch;
110     }
111     else {
112     if (n == 0)
113         printf ("c_trygetchar: end of file\n");
114     return 0xFF;
115     }
116 }
117 
118 // ================================================================
119 // c_putchar()
120 // Writes character to stdout
121 void c_putchar (uint8_t ch)
122 {
123     printf("%c",ch);
124 }
125 
126 /**
127  * Assume that both sensors have 12 bit resolution
128  * and two data registers
129  *
130  * Pressure sensor: https://cdn.sparkfun.com/datasheets/Sensors/Pressure/MPL3115A2.pdf#
        ↪ G1007342
131  * Temp sensor: https://www.sparkfun.com/datasheets/Sensors/Temperature/tmp102.pdf
132  *
133  * TODO: simplify
134  */
135 // channel -> sensor # -> val
136 uint32_t sensors[2][2];
137 uint8_t c_i2c_request (uint8_t slaveaddr, uint8_t data) {
138     static uint8_t data_reg = 0;
139     static uint8_t pointer_reg = 1;
140     static int initialized = 0;
141     static uint32_t last_update = 0;
142     static uint32_t last[2][2] = {0};
143 
144     struct timespec tp;
145     clock_gettime(CLOCK_REALTIME, &tp);
146     uint32_t t = tp.tv_sec*1000 + tp.tv_nsec/1000000;
147 
148 #ifdef SIMULATE_SENSORS
149     if (!initialized) {
150         last_update = t;
151         last[0][T] = T0;
152         last[1][T] = T0;
153         last[0][P] = P0;
154         last[1][P] = P0;
155         sensors[0][T] = last[0][T];
156         sensors[1][T] = last[1][T];
157         sensors[0][P] = last[0][P];
158         sensors[1][P] = last[1][P];
159         initialized = 1;
160     } else if (t - last_update > SENSOR_UPDATE_MS) {
161         for (int s = 0; s < 2; ++s) {
162             last[s][T] += (rand() % 3) - 1;
163             // TODO: Temp sensor resolution is -25..85C
164             // Don't stray too far from our steam table
165             last[s][T] = min(last[s][T], 300);
166             last[s][T] = max(last[s][T], 25);
167 
168             last[s][P] += (rand() % 3) - 1 + P_BIAS;
169             // Don't stray too far from our steam table
170             last[s][P] = min(last[s][P], 5775200);
171             last[s][P] = max(last[s][P], 8000);
```

```
        }
        last_update = t;
    }
    // Smooth the transitions
    sensors[0][T] = last[0][T];
    sensors[1][T] = last[1][T];
    sensors[0][P] = last[0][P];
    sensors[1][P] = last[1][P];
#endif

    if (slaveaddr & 0x1) {
        // Write request
        pointer_reg = data % 4;
        data_reg = pointer_reg;
    } else {
        // Read request, use 7bit addressing
        uint8_t dev_addr = slaveaddr >> 1;
        switch (dev_addr) {
            case TEMP_0_I2C_ADDR:
                data_reg = (uint8_t)(sensors[0][T] >> pointer_reg*8);
                break;
            case TEMP_1_I2C_ADDR:
                data_reg = (uint8_t)(sensors[1][T] >> pointer_reg*8);
                break;
            case PRESSURE_0_I2C_ADDR:
                data_reg = (uint8_t)(sensors[0][P] >> pointer_reg*8);
                break;
            case PRESSURE_1_I2C_ADDR:
                data_reg = (uint8_t)(sensors[1][P] >> pointer_reg*8);
                break;
            default:
                data_reg = 0xAA;
                break;
        }

    }
    return data_reg;
}
```

Listing F.9: C implementation of actuation_unit.

```
#include "common.h"
#include "platform.h"
#include "actuation_logic.h"

#ifdef PLATFORM_HOST
#include <stdio.h>
#else
#include "printf.h"
#endif

#define VOTE_I(_v, _i) (((_v) >> (_i)) & 0x1)

/*@requires \valid(&trip[0..2][0..3]);
  @requires \valid(&trip_test[0..2][0..3]);
  @assigns (trip[0..2][0..3]);
  @assigns (trip_test[0..2][0..3]);
*/
static int
actuation_logic_collect_trips(uint8_t logic_no, int do_test, uint8_t trip[3][4], uint8_t
    ↪ trip_test[3][4])
{
    int err = 0;
    uint8_t test_div[2];
    get_test_instrumentation(test_div);

```

```
25 |     err |= read_instrumentation_trip_signals(trip);
26 |
27 |     /*@ loop invariant 0 <= i <= NINSTR;
28 |       @ loop assigns i;
29 |       @ loop assigns trip[0..2][0..3];
30 |       @ loop assigns trip_test[0..2][0..3];
31 |     */
32 |     for (int i = 0; i < NINSTR; ++i) {
33 |         /*@ loop invariant 0 <= c <= NTRIP;
34 |           @ loop assigns c;
35 |           @ loop assigns trip[0..2][i];
36 |           @ loop assigns trip_test[0..2][i];
37 |         */
38 |         for(int c = 0; c < NTRIP; ++c) {
39 |             uint8_t test_signal = (i == test_div[0] || i == test_div[1]);
40 |             if (do_test) {
41 |                 trip_test[c][i] = (trip[c][i] & test_signal) != 0;
42 |                 trip[c][i] &= !test_signal;
43 |             } else if (!VALID(trip[c][i])) {
44 |                 trip[c][i] = 0;
45 |             }
46 |         }
47 |     }
48 |
49 |     return err;
50 | }
51 |
52 | /*@ requires \valid(&trips[0..2][0..3]);
53 |   @ requires \valid(trips + (0..2));
54 |   @ assigns \nothing;
55 | */
56 | static uint8_t
57 | actuate_device(uint8_t device, uint8_t trips[3][4], int old)
58 | {
59 |     uint8_t res = 0;
60 |     if (device == 0) {
61 |         res = Actuate_D0(trips, old);
62 |     } else {
63 |         res = Actuate_D1(trips, old);
64 |     }
65 |     DEBUG_PRINTF(("<actuation_unit.c> actuate_device: device=0x%X, old=0x%X, out=0x%X,trips
          ↪ =[\n", device, old, res));
66 |     /*@ loop assigns i; */
67 |     for (int i = 0; i < 3; ++i) {
68 |         DEBUG_PRINTF(("["));
69 |         /*@ loop assigns div; */
70 |         for (int div = 0; div < 4; ++div) {
71 |         DEBUG_PRINTF(("%u,",trips[i][div]));
72 |         }
73 |         DEBUG_PRINTF(("],"));
74 |     }
75 |     DEBUG_PRINTF(("]\n"));
76 |     return res;
77 | }
78 |
79 | /*@requires \valid(state);
80 |   @requires logic_no < NVOTE_LOGIC;
81 |   @requires device < NDEV;
82 |   @requires \valid(trip + (0..2));
83 |   @requires \valid(trip_test + (0..2));
84 |   @requires \valid(&trip[0..2][0..3]);
85 |   @requires \valid(&trip_test[0..2][0..3]);
86 |   @assigns state->vote_actuate[device];
87 |   @assigns core.test.actuation_old_vote;
88 |   @assigns core.test.test_actuation_unit_done[logic_no];
89 | */
```

```
90  static void
91  actuation_logic_vote_trips(uint8_t logic_no, int do_test, uint8_t device, uint8_t trip
        ↪ [3][4], uint8_t trip_test[3][4], struct actuation_logic *state)
92  {
93      if (do_test && get_test_device() == device) {
94          if (!is_actuation_unit_test_complete(logic_no)) {
95              set_actuation_unit_test_input_vote(logic_no, state->vote_actuate[device] != 0);
96              state->vote_actuate[device] = actuate_device(device, trip_test, state->
                    ↪ vote_actuate[device] != 0);
97          }
98      } else {
99          state->vote_actuate[device] = actuate_device(device, trip, state->vote_actuate[device
                ↪ ] != 0);
100     }
101 }
102
103 /*@ requires logic_no < NVOTE_LOGIC;
104   @ requires \valid(state);
105   @ assigns state->vote_actuate[0..1];
106
107   @ assigns core.test.actuation_old_vote;
108   @ assigns core.test.test_actuation_unit_done[logic_no];
109 */
110 static int
111 actuation_logic_vote(uint8_t logic_no, int do_test, struct actuation_logic *state)
112 {
113     int err = 0;
114     uint8_t trip[3][4];
115     uint8_t trip_test[3][4];
116
117     err = actuation_logic_collect_trips(logic_no, do_test, trip, trip_test);
118
119     actuation_logic_vote_trips(logic_no, do_test, 0, trip, trip_test, state);
120     actuation_logic_vote_trips(logic_no, do_test, 1, trip, trip_test, state);
121
122     return err;
123 }
124
125 /*@requires \valid(cmd);
126   @requires \valid(state);
127   @assigns state->manual_actuate[0..1];
128   @ensures -1 <= \result <= 0;
129 */
130 static int
131 actuation_handle_command(uint8_t logic_no, struct actuation_command *cmd, struct
        ↪ actuation_logic *state)
132 {
133     if (cmd->device <= 1)
134         state->manual_actuate[cmd->device] = cmd->on;
135     return 0;
136 }
137
138 /*@requires \valid(state);
139   @requires logic_no < NVOTE_LOGIC;
140   @assigns state->vote_actuate[0..1];
141   @assigns core.test.test_actuation_unit_done[logic_no];
142   @ensures -1 <= \result <= 0;
143 */
144 static int
145 output_actuation_signals(uint8_t logic_no, int do_test, struct actuation_logic *state)
146 {
147     int err = 0;
148
149     /*@ loop invariant 0 <= d <= NDEV;
150       @ loop invariant -1 <= err <= 0;
151       @ loop assigns d, err;
```

```
    */
    for (int d = 0; d < NDEV; ++d) {
        uint8_t on = state->vote_actuate[d] || state->manual_actuate[d];
        if (!do_test || !is_actuation_unit_test_complete(logic_no)) {
            err |= set_output_actuation_logic(logic_no, d, BIT(do_test, on));
        }
    }
    if (do_test && !is_actuation_unit_test_complete(logic_no)) {
        // Reset internal state
        state->vote_actuate[0] = 0;
        state->vote_actuate[1] = 0;
        set_actuation_unit_test_complete(logic_no, 1);
    }

    return err;
}

int actuation_unit_step(uint8_t logic_no, struct actuation_logic *state)
{
    int err = 0;
    uint8_t test_div[2];

    get_test_instrumentation(test_div);
    int do_test = logic_no == get_test_actuation_unit() &&
                is_instrumentation_test_complete(test_div[0]) &&
                is_instrumentation_test_complete(test_div[1]) &&
                is_test_running();

    if (do_test && is_actuation_unit_test_complete(logic_no))
        return 0;

    if (!do_test && is_actuation_unit_test_complete(logic_no)) {
        set_output_actuation_logic(logic_no, get_test_device(), 0);
        set_actuation_unit_test_complete(logic_no, 0);
        return 0;
    }

    /* Read trip signals & vote */
    err |= actuation_logic_vote(logic_no, do_test, state);

    /* Handle any external commands */
    struct actuation_command cmd;
    int read_cmd = read_actuation_command(logic_no, &cmd);
    if (read_cmd > 0) {
        err |= actuation_handle_command(logic_no, &cmd, state);
    } else if (read_cmd < 0) {
        err |= -read_cmd;
    }

    /* Actuate devices based on voting and commands */
    err |= output_actuation_signals(logic_no, do_test, state);
    return err;
}
```

Listing F.10: C implementation of actuator_impl.

```
typedef unsigned _ExtInt(1) w1;
typedef unsigned _ExtInt(2) w2;
w1 static rotl1(w1 x, w1 shf)
{
    w1 offset = 1;
    return x << shf | x >> offset - shf;
}
w1 static rotr1(w1 x, w1 shf)
{
    w1 offset = 1;
```

```
    return x >> shf | x << offset - shf;
}
w2 static rotl2(w2 x, w2 shf)
{
    w2 offset = 2;
    return x << shf | x >> offset - shf;
}
w2 static rotr2(w2 x, w2 shf)
{
    w2 offset = 2;
    return x >> shf | x << offset - shf;
}
w1 ActuateActuator(w2 inputs4683)
{
    w1 app_4097;
    w1 return_4096;
    return_4096 = 0;
    app_4097 = inputs4683 >> (w1) 0 & 1 | inputs4683 >> (w1) 1 & 1;
    return_4096 = app_4097;
    return return_4096;
}
```

Listing F.11: C implementation of firmware.

```
#include <stdint.h>
#include "bsp.h"
#include "printf.h"
//
// int main(void)
// {
// volatile uint32_t *gpio = (void*) GPIO_REG;
// uint32_t cnt = 0;
// //char line[256] = {0};
// //printf("Hello world\n");
// while(1) {
// //printf("%u miliseconds passed, GPIO=0x%X\n",time_in_ms(), *gpio);
// //printf("%d seconds passed...and a sensor reads 0x%X\n",time_in_s(),i2c_read(0x64, 0x0B))
    ↪ ;

// // NOTE this is still line buffered
// //uint8_t c = soc_getchar();
// //printf(">>>%c<<<\n",c);
// // for (unsigned int i = 0; i < sizeof(line); i++) {
// // line[i] = soc_getchar();
// // if (line[i] == 0 || line[i] == '\n') {
// // break;
// // }
// // }
// // printf(">>>%s<<<\n",line);
// *gpio = cnt;
// cnt++;
// cnt = cnt % 256;
// //delay_ms(1000);
// delay(100000);
// }

// return 0;
// }

int main()
{
       volatile uint32_t *gpio = (void*)GPIO_REG;
   *gpio = 0;
       uint32_t cnt = 0;
       while(1)
       {
```

```
                delay_ms(1000);
        if (cnt == 1) {
            cnt = 0;
        } else {
            cnt = 1;
        }
                *gpio = cnt;
        printf("%u miliseconds passed, GPIO=0x%X\n",time_in_ms(), *gpio);
        }
        return 0;
}
```

Listing F.12: C implementation of actuator.

```
#include "platform.h"
#include "actuate.h"
#include "actuation_logic.h"

#ifdef PLATFORM_HOST
#include <stdio.h>
#else
#include "printf.h"
#endif

#define w1 uint8_t
#define w2 uint8_t

/*@ requires \true;
  @ assigns core.test.test_device_done[0..2];
  @ assigns core.test.test_device_result[0..2];
  @ ensures \true;
*/
int actuate_devices(void)
{
  int err = 0;
  int do_test = is_test_running() && is_actuation_unit_test_complete(get_test_actuation_unit
      ↪ ());
  DEBUG_PRINTF(("<actuator.c> actuate_devices, do_test = %i\n",do_test));

  if (!do_test) {
    DEBUG_PRINTF(("<actuator.c> actuate_devices: set actuate test complete to FALSE\n"));
    set_actuate_test_complete(0, 0);
    set_actuate_test_complete(1, 0);
  }

  /*@ loop invariant 0 <= d && d <= NDEV;
    @ loop assigns d, err, core.test.test_device_done[0..2], core.test.test_device_result
        ↪ [0..2];
    */
  for (int d = 0; d < NDEV; ++d) {
    uint8_t votes = 0;
    uint8_t test_votes = 0;

    /*@ loop invariant 0 <= l && l <= NVOTE_LOGIC;
      @ loop assigns l, err, test_votes, votes;
    */
    for (int l = 0; l < NVOTE_LOGIC; ++l) {
      uint8_t this_vote = 0;
      err |= get_actuation_state(l, d, &this_vote);
      if (do_test && l == get_test_actuation_unit())
        test_votes |= ((this_vote & 0x1) << d);
      else if (VALID(this_vote))
        votes |= (this_vote << d);
    }

    if (do_test && d == get_test_device()) {
```

```
    if (!is_actuate_test_complete(get_test_device())) {
      DEBUG_PRINTF(("<actuator.c> actuate_devices: set_actuate_test_result(0x%X,
           ↪ ActuateActuator(0x%X))\n",
             d, test_votes));
      set_actuate_test_result(d, ActuateActuator(test_votes));
      set_actuate_test_complete(d, 1);
    }
  }

  // Call out to actuation policy
  DEBUG_PRINTF(("<actuator.c> actuate_devices: Call out to actuation policy,
       ↪ set_actuate_device(0x%X, ActuateActuator(0x%X))\n",
           d, votes));
  err |= set_actuate_device(d, ActuateActuator(votes));
 }

 return err;
}
```

Listing F.13: C implementation of instrumentation_generated_SystemVerilog.

```
#ifdef PLATFORM_HOST
#include "../generated/SystemVerilog/verilator/generate_sensor_trips/VGenerate_Sensor_Trips.
     ↪ h"
#include "../generated/SystemVerilog/verilator/is_ch_tripped/VIs_Ch_Tripped.h"
#include <stdio.h>
#else
#include "printf.h"
#endif

#define Generate_Sensor_Trips Generate_Sensor_Trips_generated_SystemVerilog
#define Is_Ch_Tripped Is_Ch_Tripped_generated_SystemVerilog
#define instrumentation_step instrumentation_step_generated_SystemVerilog
#include "../components/instrumentation.c"

static uint8_t lookup[8] = { 0x0, 0b100, 0b010, 0b110, 0b001, 0b101, 0b011, 0b111 };

#ifdef PLATFORM_HOST
static VIs_Ch_Tripped is_tripped;
static VGenerate_Sensor_Trips gen_trips;

uint8_t Is_Ch_Tripped(uint8_t mode, uint8_t trip)
{
    is_tripped.mode = mode;
    is_tripped.sensor_tripped = trip;
    is_tripped.eval();
    uint32_t val = (trip & 0x1) << 3| (mode & 0x3) << 1| 0x0;
    DEBUG_PRINTF(("<instrumentation_generated_SystemVerilog.c> Is_Ch_Tripped: mode=0x%X, trip
         ↪ =0x%X, base=0x%X, res=0x%X\n",
    mode, trip, val,is_tripped.out));
    return is_tripped.out;
}

uint8_t Generate_Sensor_Trips(uint32_t vals[3], uint32_t setpoints[3])
{
    gen_trips.vals[0] = vals[2];
    gen_trips.vals[1] = vals[1];
    gen_trips.vals[2] = vals[0];
    gen_trips.setpoints[0] = setpoints[2];
    gen_trips.setpoints[1] = setpoints[1];
    gen_trips.setpoints[2] = setpoints[0];
    gen_trips.eval();
    uint8_t out = gen_trips.out;
    DEBUG_PRINTF(("<instrumentation_generated_SystemVerilog.c> Generate_Sensor_Trips: vals=[%
         ↪ u,%u,%u], setpoints=[%u,%u,%u], lookup[%d]=0x%X\n",
    vals[0],vals[1],vals[2],setpoints[0],setpoints[1],setpoints[2],out,lookup[out]));
```

```c
    return lookup[out];
}
#else
#include "bsp.h"
#include "platform.h"

uint8_t Is_Ch_Tripped(uint8_t mode, uint8_t trip)
{
    // wdata[0] - fnc select ( 0 - is_channel_tripped | 1 - generate_sensor_trips)
    // wdata[2:1] - mode
    // wdata[3] - sensor_tripped
    // rg_instr_hand_res[2:0] - result
    // rg_instr_hand_res[31] - fnc select ( 0 - is_channel_tripped | 1 - generate_sensor_trips )
    uint32_t val = (trip & 0x1) << 3| (mode & 0x3) << 1| 0x0;
    write_reg(INSTRUMENTATION_GENERATED_REG_BASE, val);
    uint8_t res = (uint8_t) (read_reg(INSTRUMENTATION_GENERATED_REG_RESULT) & 0x1);
    DEBUG_PRINTF(("<instrumentation_generated_SystemVerilog.c> Is_Ch_Tripped: mode=0x%X, trip =0x%X, base=0x%X, res=0x%X\n",
    mode, trip, val,res));
    return res;
}

uint8_t Generate_Sensor_Trips(uint32_t vals[3], uint32_t setpoints[3])
{
    // Set value for setpoints
    write_reg(INSTRUMENTATION_GENERATED_REG_SETPOINT_VAL_0, setpoints[0]);
    write_reg(INSTRUMENTATION_GENERATED_REG_SETPOINT_VAL_1, setpoints[1]);
    write_reg(INSTRUMENTATION_GENERATED_REG_SETPOINT_VAL_2, setpoints[2]);
    write_reg(INSTRUMENTATION_GENERATED_REG_INSTR_VAL_0, vals[0]);
    write_reg(INSTRUMENTATION_GENERATED_REG_INSTR_VAL_1, vals[1]);
    write_reg(INSTRUMENTATION_GENERATED_REG_INSTR_VAL_2, vals[2]);
    write_reg(INSTRUMENTATION_GENERATED_REG_BASE, 0x1);
    uint8_t out = (uint8_t) (read_reg(INSTRUMENTATION_GENERATED_REG_RESULT) & 0x7);
    DEBUG_PRINTF(("<instrumentation_generated_SystemVerilog.c> Generate_Sensor_Trips: vals=[% u,%u,%u], setpoints=[%u,%u,%u], lookup[%d]=0x%X\n",
    vals[0],vals[1],vals[2],setpoints[0],setpoints[1],setpoints[2],out,lookup[out]));
    return lookup[out];
}
#endif
```

Listing F.14: C implementation of core.

```c
#include "core.h"
#include "platform.h"
#include "actuate.h"
#include "rts.h"
#include <string.h>

#ifdef PLATFORM_HOST
#include <stdio.h>
#else
#include "printf.h"
#endif

#define INST_OFFSET 0
#define ACT_OFFSET 5
char INSTR_LINE_FMT[] = "#I %d (%c): T[%10d %c %d] P[%10d %c %d] S[%10d %c %d]";
char ACT_LINE_FMT[] = "#A %d [%d %d]";

const char self_test_running[] = "SELF TEST: RUNNING";
const char self_test_not_running[] = "SELF TEST: NOT RUNNING";
const char pass[] = "LAST TEST: PASS";
const char fail[] = "LAST TEST: FAIL";

char sensor_warning[] = "WARNING: LARGE SENSOR DIFFERENTIAL";
```

```
char sensor_ok[] = "SENSORS OK";

#ifdef ENABLE_SELF_TEST
struct testcase {
  uint32_t input[4][2];
  uint32_t setpoints[4][3];
  uint8_t instrumentation[2];
  uint8_t actuation_unit;
  uint8_t device;
  uint8_t expect;
} tests[] = {
// Test data generated from Cryptol RTS::SelfTestOracleHalf
#include "self_test_data/tests.inc.c"
};
#endif

char mode_char(uint8_t mode) {
  switch (mode) {
  case BYPASS:
    return 'B';
  case OPERATE:
    return 'O';
  case TRIP:
    return 'T';
  default:
    return '?';
  }
}

char maint_char(uint8_t mode) {
  if (mode)
    return 'M';
  else
    return '_';
}

int update_ui_instr(struct ui_values *ui) {
  int err = 0;
  int sensor_differential = 0;

  char line[256];

  for (uint8_t i = 0; i < NDIVISIONS; ++i) {
    for (uint8_t ch = 0; ch < NTRIP; ++ch) {
      if ((err = get_instrumentation_value(i, ch, &ui->values[i][ch])) < 0)
        return err;
      if ((err = get_instrumentation_mode(i, ch, &ui->bypass[i][ch])) < 0)
        return err;
      if ((err = get_instrumentation_trip(i, ch, &ui->trip[i][ch])) < 0)
        return err;
    }
    if ((err = get_instrumentation_maintenance(i, &ui->maintenance[i])) < 0)
      return err;

    snprintf(line, sizeof(line), INSTR_LINE_FMT, INST_OFFSET + i,
            maint_char(ui->maintenance[i]), ui->values[i][T],
            mode_char(ui->bypass[i][T]), 0 != ui->trip[i][T], ui->values[i][P],
            mode_char(ui->bypass[i][P]), 0 != ui->trip[i][P], ui->values[i][S],
            mode_char(ui->bypass[i][S]), 0 != ui->trip[i][S]);

    set_display_line(ui, i, line, sizeof(line));
  }

  // Flag any sensor differences that exceed thresholds
  for (uint8_t i = 0; i < NDIVISIONS; ++i) {

```

```
    if (ui->maintenance[i])
      continue;

    for (uint8_t j = 0; j < NDIVISIONS; ++j) {
      if (ui->maintenance[j])
        continue;

      sensor_differential |=
        (ui->values[i][T] > ui->values[j][T] &&
         ui->values[i][T] - ui->values[j][T] > T_THRESHOLD);
      sensor_differential |=
        (ui->values[i][P] > ui->values[j][P] &&
         ui->values[i][P] - ui->values[j][P] > P_THRESHOLD);
    }
  }

  if (sensor_differential)
    set_display_line(ui, 14, sensor_warning, sizeof(sensor_warning));
  else
    set_display_line(ui, 14, sensor_ok, sizeof(sensor_ok));

  return err;
}

int update_ui_actuation(struct ui_values *ui) {
  int err = 0;
  for (int i = 0; i < 2; ++i) {
    char line[256];
    for (int d = 0; d < 2; ++d) {
      uint8_t val;
      err |= get_actuation_state(i, d, &val);
      ui->actuators[i][d] = val;
    }
    snprintf(line, sizeof(line), ACT_LINE_FMT, i, ui->actuators[i][0],
            ui->actuators[i][1]);
    set_display_line(ui, ACT_OFFSET + i, line, sizeof(line));
  }

  return err;
}

int update_ui(struct ui_values *ui) {
  DEBUG_PRINTF(("<core.c> update_ui\n"));
  int err = 0;
  err |= update_ui_instr(ui);
  err |= update_ui_actuation(ui);

  return err;
}

int set_display_line(struct ui_values *ui, uint8_t line_number, char *display, uint32_t size
    ↪ ) {
  memset(ui->display[line_number], ' ', LINELENGTH);
  strncpy(ui->display[line_number], (const char*)display, LINELENGTH);
  return 0;
}

#ifdef ENABLE_SELF_TEST
int end_test(struct test_state *test, struct ui_values *ui) {
    static int cnt = 0;
    int passed =
        test->test_device_result[test->test_device]
      == (test->self_test_expect || test->actuation_old_vote);
    test->failed = !passed;
    DEBUG_PRINTF(("<core.c> end_test #%d: test->test_device_result[%u]=0x%X\n", cnt, test->
        ↪ test_device, test->test_device_result[test->test_device]));
```

```
154     DEBUG_PRINTF(("<core.c> end_test #%d: (test->self_test_expect || test->actuation_old_vote
            ↪ )=0x%X\n", cnt, (test->self_test_expect || test->actuation_old_vote)));
155
156     // Reset state
157     set_test_running(0);
158
159     if (passed) {
160       set_display_line(ui, 16, (char*)pass, 0);
161       test->test++;
162       if (test->test >= sizeof(tests)/sizeof(struct testcase)) {
163         test->test = 0;
164         test->test_timer_start = time_in_s();
165       }
166     } else {
167       set_display_line(ui, 16, (char*)fail, 0);
168       set_display_line(ui, 20, (char*)"A TEST FAILED", 0);
169     }
170     DEBUG_PRINTF(("<core.c> end_test #%d: Passed: %d\n", cnt, passed));
171     cnt++;
172     return passed;
173 }
174
175 int components_ready() {
176   return !is_instrumentation_test_complete(0)
177         && !is_instrumentation_test_complete(1)
178         && !is_instrumentation_test_complete(2)
179         && !is_instrumentation_test_complete(3)
180         && !is_actuation_unit_test_complete(0)
181         && !is_actuation_unit_test_complete(1)
182         && !is_actuate_test_complete(0)
183         && !is_actuate_test_complete(1);
184 }
185
186 int self_test_timer_expired(struct test_state *test) {
187   uint32_t t = time_in_s();
188   uint32_t diff = t - test->test_timer_start;
189   return SELF_TEST_PERIOD_SEC < diff;
190 }
191
192 int should_start_self_test(struct test_state *test) {
193   int retval = (!is_test_running()) && (self_test_timer_expired(test) || (test->test != 0));
194   return retval;
195 }
196
197 int test_step(struct test_state *test, struct ui_values *ui) {
198   DEBUG_PRINTF(("<core.c> test_step: Has test failed? %u\n",test->failed));
199   int err = 0;
200
201   if(!test->failed && should_start_self_test(test)) {
202     if (components_ready())
203     {
204       struct testcase *next = &tests[test->test];
205       test->self_test_expect = next->expect;
206       test->test_device = next->device;
207       test->test_actuation_unit = next->actuation_unit;
208       DEBUG_PRINTF(("<core.c> test_step: starting new test. test->self_test_expect=%u,test->
              ↪ test_device=%u, test->test_actuation_unit=%u,\n",
209         test->self_test_expect,test->test_device,test->test_actuation_unit));
210       memcpy(test->test_instrumentation, next->instrumentation, 2);
211       memcpy(test->test_inputs, next->input, 2*4*sizeof(uint32_t));
212       memcpy(test->test_setpoints, next->setpoints, 3*4*sizeof(uint32_t));
213
214       set_test_running(1);
215       set_display_line(ui, 15, (char *)self_test_running, 0);
216     }
217   } else if (is_test_running() && test->test_device_done[test->test_device]) {
```

```
218     DEBUG_PRINTF(("<core.c> test_step: Ending test\n"));
219     int passed = end_test(test, ui);
220     if(!passed) err = -1;
221   } else if (!is_test_running()) {
222     DEBUG_PRINTF(("<core.c> test_step:Continuing test\n"));
223     set_display_line(ui, 15, (char *)self_test_not_running, 0);
224   } else {
225     DEBUG_PRINTF(("<core.c> test_step:Catchall\n"));
226   }
227
228   return err;
229 }
230 #endif
231
232 void core_init(struct core_state *c) {
233   c->test.test_timer_start = time_in_s();
234   c->test.failed = 0;
235 }
236
237 int core_step(struct core_state *c) {
238   int err = 0;
239   struct rts_command rts;
240 #ifndef ENABLE_SELF_TEST
241   time_in_s();
242 #endif
243
244   if (!c->error) {
245     // Actuate devices if necessary
246     int retval = actuate_devices_generated_C();
247     DEBUG_PRINTF(("<core.c> actuate_devices_generated_C: 0x%X\n", retval));
248   }
249
250   // Let's allow command processing even if an error is detected.
251   // In a real system, we would probably want to disconnect the device
252   // and perform maintenance.
253   int read_cmd = read_rts_command(&rts);
254   if (read_cmd < 0) {
255     err |= -read_cmd;
256   } else if (read_cmd > 0) {
257     switch (rts.type) {
258     case INSTRUMENTATION_COMMAND:
259       err |= send_instrumentation_command(rts.instrumentation_division,
260                                           &(rts.cmd.instrumentation));
261       break;
262
263     case ACTUATION_COMMAND:
264       err |= send_actuation_command(0, &rts.cmd.act);
265       err |= send_actuation_command(1, &rts.cmd.act);
266       break;
267
268     default:
269       break;
270     }
271   }
272
273 #ifdef ENABLE_SELF_TEST
274   err |= test_step(&c->test, &c->ui);
275 #endif
276   err |= update_ui(&c->ui);
277
278   c->error = err;
279   return err;
280 }
```

Listing F.15: C implementation of instrumentation_impl.

```c
#include <stdint.h>

// Identified by SAW: vals[2] and setpoints[2] must be less than 0x80000000
uint8_t Generate_Sensor_Trips(uint32_t vals[3], uint32_t setpoints[3])
{
    uint8_t trips_out = 0;
    trips_out |= Trip(vals, setpoints, 0);
    //@ assert trips_out <= 0x1;
    trips_out |= (Trip(vals, setpoints, 1) << 1);
    //@ assert trips_out <= 0x3;
    trips_out |= (Trip(vals, setpoints, 2) << 2);

    return trips_out;
}

uint8_t Trip(uint32_t vals[3], uint32_t setpoints[3], uint8_t ch)
{
    if (ch <= 1) {
        return (setpoints[ch] < vals[ch]);
    } else {
        return ((int32_t)vals[ch] < (int32_t)setpoints[ch]);
    }
}

uint8_t Is_Ch_Tripped(uint8_t mode, uint8_t sensor_tripped)
{
    return (mode == 2) || ((mode == 1) && sensor_tripped);
}
```

Listing F.16: C implementation of instrumentation_handwritten_C.

```c
#define Generate_Sensor_Trips Generate_Sensor_Trips_handwritten_C
#define Is_Ch_Tripped Is_Ch_Tripped_handwritten_C
#define Trip Trip_handwritten_C
#define instrumentation_step instrumentation_step_handwritten_C
#include "../components/instrumentation.c"
#include "../handwritten/C/instrumentation_impl.c"
```

Listing F.17: C implementation of instrumentation.

```c
#include "instrumentation.h"
#include "platform.h"
#include "common.h"
#include "core.h"
#include <string.h>

#define TRIP_I(_v, _i) (((_v) >> (_i)) & 0x1)

/*@requires div < NINSTR;
  @requires \valid(state);
  @requires \valid(state->reading + (0.. NTRIP-1));
  @requires \valid(state->test_reading + (0.. NTRIP-1));
  @requires \valid(state->setpoints + (0.. NTRIP-1));
  @requires \valid(state->sensor_trip + (0.. NTRIP-1));
  @assigns state->reading[0.. NTRIP-1];
  @assigns state->test_reading[0.. NTRIP-1];
  @assigns state->sensor_trip[0.. NTRIP-1];
  @ensures -1 <= \result <= 0;
*/
static int instrumentation_step_trip(uint8_t div,
                                      int do_test,
                                      struct instrumentation_state *state) {
  int err = 0;

```

```
  if (do_test) {
    err |= read_test_instrumentation_channel(div, T, &state->test_reading[T]);
    err |= read_test_instrumentation_channel(div, P, &state->test_reading[P]);
    state->test_reading[S] = Saturation(state->test_reading[T], state->test_reading[P]);
  } else {
    err |= read_instrumentation_channel(div, T, &state->reading[T]);
    err |= read_instrumentation_channel(div, P, &state->reading[P]);
    state->reading[S] = Saturation(state->reading[T], state->reading[P]);
  }

  uint8_t new_trips = 0;

  if (do_test) {
    uint32_t setpoints[3];
    err |= get_instrumentation_test_setpoints(div, &setpoints[0]);
    new_trips = Generate_Sensor_Trips(state->test_reading, setpoints);
  } else {
    new_trips = Generate_Sensor_Trips(state->reading, state->setpoints);
  }

  /*@loop invariant 0 <= i && i <= NTRIP;
    @loop assigns i;
    @loop assigns state->sensor_trip[0.. NTRIP-1];
  */
  for (int i = 0; i < NTRIP; ++i) {
    state->sensor_trip[i] = TRIP_I(new_trips, i);
  }

  return err;
}

/*@requires \valid(i_cmd);
  @requires \valid(state);
  @requires state->mode[0] \in {0,1,2};
  @requires state->mode[1] \in {0,1,2};
  @requires state->mode[2] \in {0,1,2};
  @assigns state->maintenance, state->mode[0..2], state->setpoints[0..2];
  @ensures -1 <= \result <= 0;
  @ensures state->mode[0] \in {0,1,2};
  @ensures state->mode[1] \in {0,1,2};
  @ensures state->mode[2] \in {0,1,2};
*/
static int instrumentation_handle_command(uint8_t div,
                                          struct instrumentation_command *i_cmd,
                                          struct instrumentation_state *state) {
  struct set_maintenance set_maint;
  struct set_mode set_mode;
  struct set_setpoint set_setpoint;

  switch (i_cmd->type) {
  case SET_MAINTENANCE:
    set_maint = i_cmd->cmd.maintenance;
    state->maintenance = set_maint.on;
    break;

  case SET_MODE:
    set_mode = i_cmd->cmd.mode;
    if (state->maintenance && set_mode.channel < NTRIP &&
        set_mode.mode_val < NMODES) {
      state->mode[set_mode.channel] = set_mode.mode_val;
    }
    break;

  case SET_SETPOINT:
    set_setpoint = i_cmd->cmd.setpoint;
    if (state->maintenance && set_setpoint.channel < NTRIP) {
```

```
      state->setpoints[set_setpoint.channel] = set_setpoint.val;
    }
    break;

  default:
    return -1;
  }

  return 0;
}

/*@ requires div < NINSTR;
  @ requires \valid(state);
  @ requires state->mode[0] \in {0,1,2};
  @ requires state->mode[1] \in {0,1,2};
  @ requires state->mode[2] \in {0,1,2};
  @ assigns core.test.test_instrumentation_done[div];
  @ ensures \result <= 0;
*/
static int instrumentation_set_output_trips(uint8_t div,
                                            int do_test,
                                            struct instrumentation_state *state)
{
  /*@ loop invariant 0 <= i <= NTRIP;
    @ loop assigns i;
  */
  for (int i = 0; i < NTRIP; ++i) {
    uint8_t mode = do_test ? 1 : state->mode[i];
    set_output_instrumentation_trip(div, i, BIT(do_test, Is_Ch_Tripped(mode, 0 != state->
        ↪ sensor_trip[i])));
  }

  if (do_test) {
    set_instrumentation_test_complete(div, 1);
  }

  return 0;
}

int instrumentation_step(uint8_t div, struct instrumentation_state *state) {
  int err = 0;

  uint8_t test_div[2];
  get_test_instrumentation(test_div);
  int do_test = (div == test_div[0] || div == test_div[1]) && is_test_running();

  if (do_test && is_instrumentation_test_complete(div))
    return 0;

  if (!do_test && is_instrumentation_test_complete(div)) {
    set_instrumentation_test_complete(div, 0);
  }

  /* Read trip signals & vote */
  err |= instrumentation_step_trip(div, do_test, state);

  /* Handle any external commands */
  struct instrumentation_command i_cmd;
  int read_cmd = read_instrumentation_command(div, &i_cmd);
  if (read_cmd > 0) {
    err |= instrumentation_handle_command(div, &i_cmd, state);
  } else if (read_cmd < 0) {
    err |= -read_cmd;
  }

  /* Actuate devices based on voting and commands */
```

```
  err |= instrumentation_set_output_trips(div, do_test, state);
  return err;
}
```

Listing F.18: C implementation of actuation_unit_generated_C.

```
#define Actuate_D0 Actuate_D0_generated_C
#define Actuate_D1 Actuate_D1_generated_C
#define Coincidence_2_4 Coincidence_2_4_generated_C
#define actuation_unit_step actuation_unit_step_generated_C
#include "../components/actuation_unit.c"
#include "../generated/C/actuation_unit_impl.c"
```

Listing F.19: C implementation of actuator_impl.

```
typedef unsigned _ExtInt(1) w1;
typedef unsigned _ExtInt(2) w2;
w1 static rotl1(w1 x, w1 shf)
{
    w1 offset = 1;
    return x << shf | x >> offset - shf;
}
w1 static rotr1(w1 x, w1 shf)
{
    w1 offset = 1;
    return x >> shf | x << offset - shf;
}
w2 static rotl2(w2 x, w2 shf)
{
    w2 offset = 2;
    return x << shf | x >> offset - shf;
}
w2 static rotr2(w2 x, w2 shf)
{
    w2 offset = 2;
    return x >> shf | x << offset - shf;
}
w1 ActuateActuator(w2 inputs4683)
{
    w1 app_4097;
    w1 return_4096;
    return_4096 = 0;
    app_4097 = inputs4683 >> (w1) 0 & 1 | inputs4683 >> (w1) 1 & 1;
    return_4096 = app_4097;
    return return_4096;
}
```

Listing F.20: C implementation of sense_actuate.

```
#include "common.h"
#include "platform.h"
#include "instrumentation.h"
#include "actuation_logic.h"
#include "sense_actuate.h"

#ifdef PLATFORM_HOST
#include <stdio.h>
#else
#include "printf.h"
#endif

int instrumentation_step_generated_C(uint8_t div, struct instrumentation_state *state);
int instrumentation_step_handwritten_C(uint8_t div, struct instrumentation_state *state);
```

```
int instrumentation_step_generated_SystemVerilog(uint8_t div, struct instrumentation_state *
    ↪ state);
int instrumentation_step_handwritten_SystemVerilog(uint8_t div, struct instrumentation_state
    ↪  *state);
int actuation_unit_step_generated_C(uint8_t logic_no, struct actuation_logic *state);
int actuation_unit_step_generated_SystemVerilog(uint8_t logic_no, struct actuation_logic *
    ↪ state);

int sense_actuate_init(int core_id,
                    struct instrumentation_state *instrumentation,
                    struct actuation_logic *actuation)
{
  DEBUG_PRINTF(("<sense_actuate.c> sense_actuate_init\n"));
  instrumentation_init(&instrumentation[0]);
  instrumentation_init(&instrumentation[1]);
  actuation->vote_actuate[0] = 0;
  actuation->vote_actuate[1] = 0;
  return 0;
}

int sense_actuate_step_0(struct instrumentation_state *instrumentation,
                      struct actuation_logic *actuation)
{
  int err = 0;
  err |= instrumentation_step_generated_C(0,&instrumentation[0]);
  err |= instrumentation_step_handwritten_C(1,&instrumentation[1]);
  // Do we think the devices should be actuated?
  err |= actuation_unit_step_generated_C(0,actuation);
  DEBUG_PRINTF(("<sense_actuate.c> sense_actuate_step_0, err=0x%X\n",err));
  return err;
}

int sense_actuate_step_1(struct instrumentation_state *instrumentation,
                      struct actuation_logic *actuation)
{
  int err = 0;
  err |= instrumentation_step_handwritten_SystemVerilog(2,&instrumentation[0]);
  err |= instrumentation_step_generated_SystemVerilog(3,&instrumentation[1]);
  // Do we think the devices should be actuated?
  err |= actuation_unit_step_generated_SystemVerilog(1,actuation);
  DEBUG_PRINTF(("<sense_actuate.c> sense_actuate_step_1, err=0x%X\n",err));
  return err;
}
```

Listing F.21: C implementation of instrumentation_generated_C.

```
#define Generate_Sensor_Trips Generate_Sensor_Trips_generated_C
#define Is_Ch_Tripped Is_Ch_Tripped_generated_C
#define Trip Trip_generated_C
#define instrumentation_step instrumentation_step_generated_C
#include "../components/instrumentation.c"
#include "../generated/C/instrumentation_impl.c"
```

Listing F.22: C implementation of instrumentation_impl.

```
#include <stdint.h>

// Identified by SAW: vals[2] and setpoints[2] must be less than 0x80000000
uint8_t Generate_Sensor_Trips(uint32_t vals[3], uint32_t setpoints[3])
{
    uint8_t trips_out = 0;
    trips_out |= Trip(vals, setpoints, 0);
    //@ assert trips_out <= 0x1;
    trips_out |= (Trip(vals, setpoints, 1) << 1);
    //@ assert trips_out <= 0x3;
```

```
    trips_out |= (Trip(vals, setpoints, 2) << 2);

    return trips_out;
}

uint8_t Trip(uint32_t vals[3], uint32_t setpoints[3], uint8_t ch)
{
    if (ch <= 1) {
        return (setpoints[ch] < vals[ch]);
    } else {
        return ((int32_t)vals[ch] < (int32_t)setpoints[ch]);
    }
}

uint8_t Is_Ch_Tripped(uint8_t mode, uint8_t sensor_tripped)
{
    return (mode == 2) || ((mode == 1) && sensor_tripped);
}
```

Listing F.23: C implementation of instrumentation_impl.

```
#include <stdint.h>

// Identified by SAW: vals[2] and setpoints[2] must be less than 0x80000000
uint8_t Generate_Sensor_Trips(uint32_t vals[3], uint32_t setpoints[3])
{
    uint8_t trips_out = 0;
    trips_out |= Trip(vals, setpoints, 0);
    //@ assert trips_out <= 0x1;
    trips_out |= (Trip(vals, setpoints, 1) << 1);
    //@ assert trips_out <= 0x3;
    trips_out |= (Trip(vals, setpoints, 2) << 2);

    return trips_out;
}

uint8_t Trip(uint32_t vals[3], uint32_t setpoints[3], uint8_t ch)
{
    if (ch <= 1) {
        return (setpoints[ch] < vals[ch]);
    } else {
        return ((int32_t)vals[ch] < (int32_t)setpoints[ch]);
    }
}

uint8_t Is_Ch_Tripped(uint8_t mode, uint8_t sensor_tripped)
{
    return (mode == 2) || ((mode == 1) && sensor_tripped);
}
```

Listing F.24: C implementation of bsp.

```
#include "bsp.h"
#include "printf.h"

uint32_t i2c_read(uint8_t addr, uint32_t data_tx)
{
  volatile uint32_t *i2c_addr = (void*) I2C_REG_ADDR;
  volatile uint32_t *i2c_data = (void*) I2C_REG_DATA;
  volatile uint32_t *i2c_status = (void*) I2C_REG_STATUS;

  // First set data
  *i2c_data = data_tx;
  // Second set address
```

```
  *i2c_addr = addr;
  // Now the transaction is initialized, so wait for completion
  delay(100);
  while (1) {
    uint32_t status = *i2c_status;
    if (status) {
      // Return the acquired data
      uint32_t data = *i2c_data;
      return data;
    }
    delay(1000);
  }
}

uint32_t time_in_s(void)
{
  uint32_t t_s = time_in_ms()/1000;
  return t_s;
}

uint32_t time_in_ms(void)
{
  volatile uint32_t *tick_reg_low = (void*) TICK_REG_LOW;
  volatile uint32_t *tick_reg_high = (void*) TICK_REG_HIGH;
  uint64_t ticks = (uint64_t) (*tick_reg_high << 31 | *tick_reg_low);
  return (uint32_t)((ticks*1000)/(CORE_FREQ*TICKS_TO_MS_MULTIPLIER));
}

void delay_ms(uint32_t ms)
{
    uint32_t ticks = ms*CORE_FREQ/1000;
    delay(ticks);
}

void delay(uint32_t count)
{
       while(count-->0) {
              __asm__ volatile ("nop");
       }
}

uint8_t soc_getchar(void)
{
  volatile uint32_t *data_rdy = (void*) UART_REG_DATA_READY;
  volatile uint32_t *rx_data = (void*) UART_REG_RX;
  int startime = time_in_ms();
  int delay_ms = 0;
  // Wait 1s for each character
  while ((delay_ms < 2000)) {
    if (*data_rdy){
        return (uint8_t)(*rx_data);
    }
    delay_ms = time_in_ms() - startime;
  }
  return 0;
}

// Read from a register
uint32_t read_reg(uint32_t reg)
{
  uint32_t *p = (void*)reg;
  return *p;
}

// Write `val` to `reg`
void write_reg(uint32_t reg, uint32_t val)
```

```
{
  uint32_t *p = (void*)reg;
  *p = val;
}
```

Listing F.25: C implementation of instrumentation_impl.

```
#include <stdint.h>

// Identified by SAW: vals[2] and setpoints[2] must be less than 0x80000000
uint8_t Generate_Sensor_Trips(uint32_t vals[3], uint32_t setpoints[3])
{
    uint8_t trips_out = 0;
    trips_out |= Trip(vals, setpoints, 0);
    //@ assert trips_out <= 0x1;
    trips_out |= (Trip(vals, setpoints, 1) << 1);
    //@ assert trips_out <= 0x3;
    trips_out |= (Trip(vals, setpoints, 2) << 2);

    return trips_out;
}

uint8_t Trip(uint32_t vals[3], uint32_t setpoints[3], uint8_t ch)
{
    if (ch <= 1) {
        return (setpoints[ch] < vals[ch]);
    } else {
        return ((int32_t)vals[ch] < (int32_t)setpoints[ch]);
    }
}

uint8_t Is_Ch_Tripped(uint8_t mode, uint8_t sensor_tripped)
{
    return (mode == 2) || ((mode == 1) && sensor_tripped);
}
```

Listing F.26: C implementation of firmware.

```
#include <stdint.h>
#include "bsp.h"
#include "printf.h"
//
// int main(void)
// {
// volatile uint32_t *gpio = (void*) GPIO_REG;
// uint32_t cnt = 0;
// //char line[256] = {0};
// //printf("Hello world\n");
// while(1) {
// //printf("%u miliseconds passed, GPIO=0x%X\n",time_in_ms(), *gpio);
// //printf("%d seconds passed...and a sensor reads 0x%X\n",time_in_s(),i2c_read(0x64, 0x0B))
    ↪ ;

// // NOTE this is still line buffered
// //uint8_t c = soc_getchar();
// //printf(">>>%c<<<\n",c);
// // for (unsigned int i = 0; i < sizeof(line); i++) {
// // line[i] = soc_getchar();
// // if (line[i] == 0 || line[i] == '\n') {
// // break;
// // }
// // }
// // printf(">>>%s<<<\n",line);
// *gpio = cnt;
// cnt++;
```

```
// cnt = cnt % 256;
// //delay_ms(1000);
// delay(100000);
// }

// return 0;
// }

int main()
{
       volatile uint32_t *gpio = (void*)GPIO_REG;
   *gpio = 0;
       uint32_t cnt = 0;
       while(1)
       {
              delay_ms(1000);
       if (cnt == 1) {
           cnt = 0;
       } else {
           cnt = 1;
       }
              *gpio = cnt;
       printf("%u miliseconds passed, GPIO=0x%X\n",time_in_ms(), *gpio);
       }
       return 0;
}
```

Listing F.27: C implementation of saturation_impl.

```
typedef unsigned _ExtInt(1) w1;
typedef unsigned _ExtInt(32) w32;
w1 static rotl1(w1 x, w1 shf)
{
    w1 offset = 1;
    return x << shf | x >> offset - shf;
}
w1 static rotr1(w1 x, w1 shf)
{
    w1 offset = 1;
    return x >> shf | x << offset - shf;
}
w32 static rotl32(w32 x, w32 shf)
{
    w32 offset = 32;
    return x << shf | x >> offset - shf;
}
w32 static rotr32(w32 x, w32 shf)
{
    w32 offset = 32;
    return x >> shf | x << offset - shf;
}
w32 Saturation(w32 t4762, w32 p4763)
{
    w32 app_4100;
    w32 app_4101;
    w1 app_4157;
    w1 app_4158;
    w1 app_4159;
    w32 app_4160;
    w32 ifv_4099;
    w32 ifv_4156;
    w32 return_4097;
    w32 table_4102[52];
    w32 v4804_4103;
    w32 v4805_4104;
    w32 v4806_4105;
```

```
    w32 v4807_4106;
    w32 v4808_4107;
    w32 v4809_4108;
    w32 v4810_4109;
    w32 v4811_4110;
    w32 v4812_4111;
    w32 v4813_4112;
    w32 v4814_4113;
    w32 v4815_4114;
    w32 v4816_4115;
    w32 v4817_4116;
    w32 v4818_4117;
    w32 v4819_4118;
    w32 v4820_4119;
    w32 v4821_4120;
    w32 v4822_4121;
    w32 v4823_4122;
    w32 v4824_4123;
    w32 v4825_4124;
    w32 v4826_4125;
    w32 v4827_4126;
    w32 v4828_4127;
    w32 v4829_4128;
    w32 v4830_4129;
    w32 v4831_4130;
    w32 v4832_4131;
    w32 v4833_4132;
    w32 v4834_4133;
    w32 v4835_4134;
    w32 v4836_4135;
    w32 v4837_4136;
    w32 v4838_4137;
    w32 v4839_4138;
    w32 v4840_4139;
    w32 v4841_4140;
    w32 v4842_4141;
    w32 v4843_4142;
    w32 v4844_4143;
    w32 v4845_4144;
    w32 v4846_4145;
    w32 v4847_4146;
    w32 v4848_4147;
    w32 v4849_4148;
    w32 v4850_4149;
    w32 v4851_4150;
    w32 v4852_4151;
    w32 v4853_4152;
    w32 v4854_4153;
    w32 v4855_4154;
    return_4097 = 0;
    app_4158 = (signed _ExtInt(32)) t4762 < (signed _ExtInt(32)) (w32) 35;
    if (app_4158)
    {
        ifv_4099 = (w32) 0;
    }
    else
    {
        app_4100 = t4762 - (w32) 35;
        app_4101 = app_4100 / (w32) 5;
        ifv_4099 = app_4101;
    }
    app_4159 = ifv_4099 < (w32) 52;
    if (app_4159)
    {
        app_4157 = (signed _ExtInt(32)) t4762 < (signed _ExtInt(32)) (w32) 35;
        if (app_4157)
```

```
        {
            ifv_4099 = (w32) 0;
        }
        else
        {
            app_4100 = t4762 - (w32) 35;
            app_4101 = app_4100 / (w32) 5;
            ifv_4099 = app_4101;
        }
        v4804_4103 = (w32) 9998;
        v4805_4104 = (w32) 12163;
        v4806_4105 = (w32) 14753;
        v4807_4106 = (w32) 17796;
        v4808_4107 = (w32) 21404;
        v4809_4108 = (w32) 25611;
        v4810_4109 = (w32) 30562;
        v4811_4110 = (w32) 36292;
        v4812_4111 = (w32) 42985;
        v4813_4112 = (w32) 50683;
        v4814_4113 = (w32) 59610;
        v4815_4114 = (w32) 69813;
        v4816_4115 = (w32) 81567;
        v4817_4116 = (w32) 94924;
        v4818_4117 = (w32) 110218;
        v4819_4118 = (w32) 127500;
        v4820_4119 = (w32) 147160;
        v4821_4120 = (w32) 169270;
        v4822_4121 = (w32) 194350;
        v4823_4122 = (w32) 222300;
        v4824_4123 = (w32) 253820;
        v4825_4124 = (w32) 288920;
        v4826_4125 = (w32) 328250;
        v4827_4126 = (w32) 371840;
        v4828_4127 = (w32) 420470;
        v4829_4128 = (w32) 474140;
        v4830_4129 = (w32) 533740;
        v4831_4130 = (w32) 599260;
        v4832_4131 = (w32) 671730;
        v4833_4132 = (w32) 751100;
        v4834_4133 = (w32) 838550;
        v4835_4134 = (w32) 934000;
        v4836_4135 = (w32) 1038600;
        v4837_4136 = (w32) 1152600;
        v4838_4137 = (w32) 1277600;
        v4839_4138 = (w32) 1413200;
        v4840_4139 = (w32) 1469600;
        v4841_4140 = (w32) 1718600;
        v4842_4141 = (w32) 1892100;
        v4843_4142 = (w32) 2079100;
        v4844_4143 = (w32) 2280400;
        v4845_4144 = (w32) 2496800;
        v4846_4145 = (w32) 2731900;
        v4847_4146 = (w32) 2984000;
        v4848_4147 = (w32) 3253900;
        v4849_4148 = (w32) 3542700;
        v4850_4149 = (w32) 3854600;
        v4851_4150 = (w32) 4187500;
        v4852_4151 = (w32) 4542300;
        v4853_4152 = (w32) 4920000;
        v4854_4153 = (w32) 5325900;
        v4855_4154 = (w32) 5775200;
        table_4102[0] = v4804_4103;
        table_4102[1] = v4805_4104;
        table_4102[2] = v4806_4105;
        table_4102[3] = v4807_4106;
        table_4102[4] = v4808_4107;
```

```
        table_4102[5] = v4809_4108;
        table_4102[6] = v4810_4109;
        table_4102[7] = v4811_4110;
        table_4102[8] = v4812_4111;
        table_4102[9] = v4813_4112;
        table_4102[10] = v4814_4113;
        table_4102[11] = v4815_4114;
        table_4102[12] = v4816_4115;
        table_4102[13] = v4817_4116;
        table_4102[14] = v4818_4117;
        table_4102[15] = v4819_4118;
        table_4102[16] = v4820_4119;
        table_4102[17] = v4821_4120;
        table_4102[18] = v4822_4121;
        table_4102[19] = v4823_4122;
        table_4102[20] = v4824_4123;
        table_4102[21] = v4825_4124;
        table_4102[22] = v4826_4125;
        table_4102[23] = v4827_4126;
        table_4102[24] = v4828_4127;
        table_4102[25] = v4829_4128;
        table_4102[26] = v4830_4129;
        table_4102[27] = v4831_4130;
        table_4102[28] = v4832_4131;
        table_4102[29] = v4833_4132;
        table_4102[30] = v4834_4133;
        table_4102[31] = v4835_4134;
        table_4102[32] = v4836_4135;
        table_4102[33] = v4837_4136;
        table_4102[34] = v4838_4137;
        table_4102[35] = v4839_4138;
        table_4102[36] = v4840_4139;
        table_4102[37] = v4841_4140;
        table_4102[38] = v4842_4141;
        table_4102[39] = v4843_4142;
        table_4102[40] = v4844_4143;
        table_4102[41] = v4845_4144;
        table_4102[42] = v4846_4145;
        table_4102[43] = v4847_4146;
        table_4102[44] = v4848_4147;
        table_4102[45] = v4849_4148;
        table_4102[46] = v4850_4149;
        table_4102[47] = v4851_4150;
        table_4102[48] = v4852_4151;
        table_4102[49] = v4853_4152;
        table_4102[50] = v4854_4153;
        table_4102[51] = v4855_4154;
        ifv_4156 = table_4102[ifv_4099];
    }
    else
    {
        ifv_4156 = (w32) 5775200;
    }
    app_4160 = p4763 - ifv_4156;
    return_4097 = app_4160;
    return return_4097;
}
```

Listing F.28: C implementation of instrumentation_handwritten_SystemVerilog.

```
#ifdef PLATFORM_HOST
#include "../handwritten/SystemVerilog/verilator/generate_sensor_trips/
    ↪ VGenerate_Sensor_Trips.h"
#include "../handwritten/SystemVerilog/verilator/is_ch_tripped/VIs_Ch_Tripped.h"
#include <stdio.h>
#else
```

```
6  #include "printf.h"
7  #endif
8
9  #define Generate_Sensor_Trips Generate_Sensor_Trips_handwritten_SystemVerilog
10 #define Is_Ch_Tripped Is_Ch_Tripped_handwritten_SystemVerilog
11 #define instrumentation_step instrumentation_step_handwritten_SystemVerilog
12 #include "../components/instrumentation.c"
13
14 static uint8_t lookup[8] = { 0x0, 0b100, 0b010, 0b110, 0b001, 0b101, 0b011, 0b111 };
15
16 #ifdef PLATFORM_HOST
17 static VIs_Ch_Tripped is_tripped;
18 static VGenerate_Sensor_Trips gen_trips;
19
20 uint8_t Is_Ch_Tripped(uint8_t mode, uint8_t trip)
21 {
22     is_tripped.mode = mode;
23     is_tripped.sensor_tripped = trip;
24     is_tripped.eval();
25     uint32_t val = (trip & 0x1) << 3| (mode & 0x3) << 1| 0x0;
26     DEBUG_PRINTF(("<instrumentation_handwritten_SystemVerilog.c> Is_Ch_Tripped: mode=0x%X,
           ↪ trip=0x%X, base=0x%X, res=0x%X\n",
27     mode, trip, val,is_tripped.out));
28     return is_tripped.out;
29 }
30
31 uint8_t Generate_Sensor_Trips(uint32_t vals[3], uint32_t setpoints[3])
32 {
33     gen_trips.vals[0] = vals[2];
34     gen_trips.vals[1] = vals[1];
35     gen_trips.vals[2] = vals[0];
36     gen_trips.setpoints[0] = setpoints[2];
37     gen_trips.setpoints[1] = setpoints[1];
38     gen_trips.setpoints[2] = setpoints[0];
39     gen_trips.eval();
40     uint8_t out = gen_trips.out;
41     DEBUG_PRINTF(("<instrumentation_handwritten_SystemVerilog.c> Generate_Sensor_Trips: vals
           ↪ =[%u,%u,%u], setpoints=[%u,%u,%u], lookup[%d]=0x%X\n",
42     vals[0],vals[1],vals[2],setpoints[0],setpoints[1],setpoints[2],out,lookup[out]));
43     return lookup[out];
44 }
45 #else
46 #include "bsp.h"
47 #include "platform.h"
48
49 uint8_t Is_Ch_Tripped(uint8_t mode, uint8_t trip)
50 {
51     // wdata[0] - fnc select ( 0 - is_channel_tripped | 1 - generate_sensor_trips)
52     // wdata[2:1] - mode
53     // wdata[3] - sensor_tripped
54     // rg_instr_hand_res[2:0] - result
55     // rg_instr_hand_res[31] - fnc select ( 0 - is_channel_tripped | 1 - generate_sensor_trips
           ↪ )
56     uint32_t val = (trip & 0x1) << 3| (mode & 0x3) << 1| 0x0;
57     write_reg(INSTRUMENTATION_HANDWRITTEN_REG_BASE, val);
58     uint8_t res = (uint8_t)(read_reg(INSTRUMENTATION_HANDWRITTEN_REG_RESULT) & 0x1);
59     DEBUG_PRINTF(("<instrumentation_handwritten_SystemVerilog.c> Is_Ch_Tripped: mode=0x%X,
           ↪ trip=0x%X, base=0x%X, res=0x%X\n",
60     mode, trip, val,res));
61     return res;
62 }
63
64 uint8_t Generate_Sensor_Trips(uint32_t vals[3], uint32_t setpoints[3])
65 {
66     // Set value for setpoints
67     write_reg(INSTRUMENTATION_HANDWRITTEN_REG_SETPOINT_VAL_0, setpoints[0]);
```

```
    write_reg(INSTRUMENTATION_HANDWRITTEN_REG_SETPOINT_VAL_1, setpoints[1]);
    write_reg(INSTRUMENTATION_HANDWRITTEN_REG_SETPOINT_VAL_2, setpoints[2]);
    write_reg(INSTRUMENTATION_HANDWRITTEN_REG_INSTR_VAL_0, vals[0]);
    write_reg(INSTRUMENTATION_HANDWRITTEN_REG_INSTR_VAL_1, vals[1]);
    write_reg(INSTRUMENTATION_HANDWRITTEN_REG_INSTR_VAL_2, vals[2]);
    write_reg(INSTRUMENTATION_HANDWRITTEN_REG_BASE, 0x1);
    uint8_t out = (uint8_t)(read_reg(INSTRUMENTATION_HANDWRITTEN_REG_RESULT) & 0x7);
    DEBUG_PRINTF(("<instrumentation_handwritten_SystemVerilog.c> Generate_Sensor_Trips: vals
        ↪ =[%u,%u,%u], setpoints=[%u,%u,%u], lookup[%d]=0x%X\n"
    ,vals[0],vals[1],vals[2],setpoints[0],setpoints[1],setpoints[2],out,lookup[out]));
    return lookup[out];
}

#endif
```

Listing F.29: C implementation of bottom.

```
#include "actuate.h"
#include "actuation_logic.h"
#include "common.h"
#include "core.h"
#include "instrumentation.h"
#include "platform.h"
#include "sense_actuate.h"

#include <assert.h>

int actuate_devices(void) {
  assert(0);
  return 0;
}

uint8_t ActuateActuator(uint8_t vs) {
  assert(0);
  return 0;
}

uint8_t Coincidence_2_4(uint8_t trips[4]) {
  assert(0);
  return 0;
}

uint8_t Actuate_D0(uint8_t trips[3][4], uint8_t old) {
  assert(0);
  return 0;
}

uint8_t Actuate_D1(uint8_t trips[3][4], uint8_t old) {
  assert(0);
  return 0;
}

int actuation_unit_step(uint8_t logic_no, struct actuation_logic *state) {
  assert(0);
  return 0;
}

int sense_actuate_init(int core_id,
                       struct instrumentation_state *instrumentation,
                       struct actuation_logic *actuation) {
  assert(0);
  return 0;
}

int sense_actuate_step_0(struct instrumentation_state *instrumentation,
                         struct actuation_logic *actuation)
```

```
50 |
51 | {
52 |   assert(0);
53 |   return 0;
54 | }
55 |
56 | int sense_actuate_step_1(struct instrumentation_state *instrumentation,
57 |                          struct actuation_logic *actuation) {
58 |   assert(0);
59 |   return 0;
60 | }
61 |
62 | void core_init(struct core_state *core) { assert(0); }
63 |
64 | int core_step(struct core_state *core) {
65 |   assert(0);
66 |   return 0;
67 | }
68 |
69 | void instrumentation_init(struct instrumentation_state *state) { assert(0); }
70 |
71 | int instrumentation_step(uint8_t div, struct instrumentation_state *state) {
72 |   assert(0);
73 |   return 0;
74 | }
75 | int read_instrumentation_channel(uint8_t div, uint8_t channel, uint32_t *val) {
76 |   assert(0);
77 |   return 0;
78 | }
79 |
80 | int get_instrumentation_value(uint8_t division, uint8_t ch, uint32_t *value) {
81 |   assert(0);
82 |   return 0;
83 | }
84 |
85 | int get_instrumentation_trip(uint8_t division, uint8_t ch, uint8_t *value) {
86 |   assert(0);
87 |   return 0;
88 | }
89 |
90 | int get_instrumentation_mode(uint8_t division, uint8_t ch, uint8_t *value) {
91 |   assert(0);
92 |   return 0;
93 | }
94 |
95 | int get_instrumentation_maintenance(uint8_t division, uint8_t *value) {
96 |   assert(0);
97 |   return 0;
98 | }
99 |
100 | int get_actuation_state(uint8_t i, uint8_t device, uint8_t *value) {
101 |   assert(0);
102 |   return 0;
103 | }
104 |
105 | int read_instrumentation_trip_signals(uint8_t arr[3][4]) {
106 |   assert(0);
107 |   return 0;
108 | }
109 |
110 | int set_output_actuation_logic(uint8_t logic_no, uint8_t device_no,
111 |                                uint8_t on) {
112 |   assert(0);
113 |   return 0;
114 | }
115 |
```

```
int set_output_instrumentation_trip(uint8_t division, uint8_t channel,
                                    uint8_t val) {
  assert(0);
  return 0;
}

int set_actuate_device(uint8_t device_no, uint8_t on) {
  assert(0);
  return 0;
}

int read_rts_command(struct rts_command *cmd) {
  assert(0);
  return 0;
}

int read_instrumentation_command(uint8_t division,
                                 struct instrumentation_command *cmd) {
  assert(0);
  return 0;
}

int send_instrumentation_command(uint8_t division,
                                 struct instrumentation_command *cmd) {
  assert(0);
  return 0;
}

int read_actuation_command(uint8_t id, struct actuation_command *cmd) {
  assert(0);
  return 0;
}

int send_actuation_command(uint8_t actuator, struct actuation_command *cmd) {
  assert(0);
  return 0;
}

int set_display_line(uint8_t line_number, const char *display, uint32_t size) {
  assert(0);
  return 0;
}

uint8_t is_test_running() {
  assert(0);
  return 0;
}

void set_test_running(int val) { assert(0); }

uint8_t get_test_device() {
  assert(0);
  return 0;
}

void get_test_instrumentation(uint8_t *id) { assert(0); }

int get_instrumentation_test_setpoints(uint8_t id, uint32_t *setpoints) {
  assert(0);
  return 0;
}
void set_instrumentation_test_complete(uint8_t div, int v) { assert(0); }
int is_instrumentation_test_complete(uint8_t id) {
  assert(0);
  return 0;
}
```

```
int read_test_instrumentation_channel(uint8_t div, uint8_t channel,
                                      uint32_t *val) {
  assert(0);
  return 0;
}

uint8_t get_test_actuation_unit() {
  assert(0);
  return 0;
}
int is_actuation_unit_under_test(uint8_t id) {
  assert(0);
  return 0;
}
void set_actuation_unit_test_complete(uint8_t div, int v) { assert(0); }
void set_actuation_unit_test_input_vote(uint8_t id, int v) { assert(0); }
int is_actuation_unit_test_complete(uint8_t id) {
  assert(0);
  return 0;
}

void set_actuate_test_result(uint8_t dev, uint8_t result) { assert(0); }
void set_actuate_test_complete(uint8_t dev, int v) { assert(0); }
int is_actuate_test_complete(uint8_t dev) {
  assert(0);
  return 0;
}

int main(int argc, char **argv) {
  assert(0);
  return 0;
}
```

Listing F.30: C implementation of saturation_impl.

```
typedef unsigned _ExtInt(1) w1;
typedef unsigned _ExtInt(32) w32;
w1 static rotl1(w1 x, w1 shf)
{
    w1 offset = 1;
    return x << shf | x >> offset - shf;
}
w1 static rotr1(w1 x, w1 shf)
{
    w1 offset = 1;
    return x >> shf | x << offset - shf;
}
w32 static rotl32(w32 x, w32 shf)
{
    w32 offset = 32;
    return x << shf | x >> offset - shf;
}
w32 static rotr32(w32 x, w32 shf)
{
    w32 offset = 32;
    return x >> shf | x << offset - shf;
}
w32 Saturation(w32 t4762, w32 p4763)
{
    w32 app_4100;
    w32 app_4101;
    w1 app_4157;
    w1 app_4158;
    w1 app_4159;
    w32 app_4160;
    w32 ifv_4099;
```

```
    w32 ifv_4156;
    w32 return_4097;
    w32 table_4102[52];
    w32 v4804_4103;
    w32 v4805_4104;
    w32 v4806_4105;
    w32 v4807_4106;
    w32 v4808_4107;
    w32 v4809_4108;
    w32 v4810_4109;
    w32 v4811_4110;
    w32 v4812_4111;
    w32 v4813_4112;
    w32 v4814_4113;
    w32 v4815_4114;
    w32 v4816_4115;
    w32 v4817_4116;
    w32 v4818_4117;
    w32 v4819_4118;
    w32 v4820_4119;
    w32 v4821_4120;
    w32 v4822_4121;
    w32 v4823_4122;
    w32 v4824_4123;
    w32 v4825_4124;
    w32 v4826_4125;
    w32 v4827_4126;
    w32 v4828_4127;
    w32 v4829_4128;
    w32 v4830_4129;
    w32 v4831_4130;
    w32 v4832_4131;
    w32 v4833_4132;
    w32 v4834_4133;
    w32 v4835_4134;
    w32 v4836_4135;
    w32 v4837_4136;
    w32 v4838_4137;
    w32 v4839_4138;
    w32 v4840_4139;
    w32 v4841_4140;
    w32 v4842_4141;
    w32 v4843_4142;
    w32 v4844_4143;
    w32 v4845_4144;
    w32 v4846_4145;
    w32 v4847_4146;
    w32 v4848_4147;
    w32 v4849_4148;
    w32 v4850_4149;
    w32 v4851_4150;
    w32 v4852_4151;
    w32 v4853_4152;
    w32 v4854_4153;
    w32 v4855_4154;
    return_4097 = 0;
    app_4158 = (signed _ExtInt(32)) t4762 < (signed _ExtInt(32)) (w32) 35;
    if (app_4158)
    {
        ifv_4099 = (w32) 0;
    }
    else
    {
        app_4100 = t4762 - (w32) 35;
        app_4101 = app_4100 / (w32) 5;
        ifv_4099 = app_4101;
```

```
    }
    app_4159 = ifv_4099 < (w32) 52;
    if (app_4159)
    {
        app_4157 = (signed _ExtInt(32)) t4762 < (signed _ExtInt(32)) (w32) 35;
        if (app_4157)
        {
            ifv_4099 = (w32) 0;
        }
        else
        {
            app_4100 = t4762 - (w32) 35;
            app_4101 = app_4100 / (w32) 5;
            ifv_4099 = app_4101;
        }
        v4804_4103 = (w32) 9998;
        v4805_4104 = (w32) 12163;
        v4806_4105 = (w32) 14753;
        v4807_4106 = (w32) 17796;
        v4808_4107 = (w32) 21404;
        v4809_4108 = (w32) 25611;
        v4810_4109 = (w32) 30562;
        v4811_4110 = (w32) 36292;
        v4812_4111 = (w32) 42985;
        v4813_4112 = (w32) 50683;
        v4814_4113 = (w32) 59610;
        v4815_4114 = (w32) 69813;
        v4816_4115 = (w32) 81567;
        v4817_4116 = (w32) 94924;
        v4818_4117 = (w32) 110218;
        v4819_4118 = (w32) 127500;
        v4820_4119 = (w32) 147160;
        v4821_4120 = (w32) 169270;
        v4822_4121 = (w32) 194350;
        v4823_4122 = (w32) 222300;
        v4824_4123 = (w32) 253820;
        v4825_4124 = (w32) 288920;
        v4826_4125 = (w32) 328250;
        v4827_4126 = (w32) 371840;
        v4828_4127 = (w32) 420470;
        v4829_4128 = (w32) 474140;
        v4830_4129 = (w32) 533740;
        v4831_4130 = (w32) 599260;
        v4832_4131 = (w32) 671730;
        v4833_4132 = (w32) 751100;
        v4834_4133 = (w32) 838550;
        v4835_4134 = (w32) 934000;
        v4836_4135 = (w32) 1038600;
        v4837_4136 = (w32) 1152600;
        v4838_4137 = (w32) 1277600;
        v4839_4138 = (w32) 1413200;
        v4840_4139 = (w32) 1469600;
        v4841_4140 = (w32) 1718600;
        v4842_4141 = (w32) 1892100;
        v4843_4142 = (w32) 2079100;
        v4844_4143 = (w32) 2280400;
        v4845_4144 = (w32) 2496800;
        v4846_4145 = (w32) 2731900;
        v4847_4146 = (w32) 2984000;
        v4848_4147 = (w32) 3253900;
        v4849_4148 = (w32) 3542700;
        v4850_4149 = (w32) 3854600;
        v4851_4150 = (w32) 4187500;
        v4852_4151 = (w32) 4542300;
        v4853_4152 = (w32) 4920000;
        v4854_4153 = (w32) 5325900;
```

```
        v4855_4154 = (w32) 5775200;
        table_4102[0] = v4804_4103;
        table_4102[1] = v4805_4104;
        table_4102[2] = v4806_4105;
        table_4102[3] = v4807_4106;
        table_4102[4] = v4808_4107;
        table_4102[5] = v4809_4108;
        table_4102[6] = v4810_4109;
        table_4102[7] = v4811_4110;
        table_4102[8] = v4812_4111;
        table_4102[9] = v4813_4112;
        table_4102[10] = v4814_4113;
        table_4102[11] = v4815_4114;
        table_4102[12] = v4816_4115;
        table_4102[13] = v4817_4116;
        table_4102[14] = v4818_4117;
        table_4102[15] = v4819_4118;
        table_4102[16] = v4820_4119;
        table_4102[17] = v4821_4120;
        table_4102[18] = v4822_4121;
        table_4102[19] = v4823_4122;
        table_4102[20] = v4824_4123;
        table_4102[21] = v4825_4124;
        table_4102[22] = v4826_4125;
        table_4102[23] = v4827_4126;
        table_4102[24] = v4828_4127;
        table_4102[25] = v4829_4128;
        table_4102[26] = v4830_4129;
        table_4102[27] = v4831_4130;
        table_4102[28] = v4832_4131;
        table_4102[29] = v4833_4132;
        table_4102[30] = v4834_4133;
        table_4102[31] = v4835_4134;
        table_4102[32] = v4836_4135;
        table_4102[33] = v4837_4136;
        table_4102[34] = v4838_4137;
        table_4102[35] = v4839_4138;
        table_4102[36] = v4840_4139;
        table_4102[37] = v4841_4140;
        table_4102[38] = v4842_4141;
        table_4102[39] = v4843_4142;
        table_4102[40] = v4844_4143;
        table_4102[41] = v4845_4144;
        table_4102[42] = v4846_4145;
        table_4102[43] = v4847_4146;
        table_4102[44] = v4848_4147;
        table_4102[45] = v4849_4148;
        table_4102[46] = v4850_4149;
        table_4102[47] = v4851_4150;
        table_4102[48] = v4852_4151;
        table_4102[49] = v4853_4152;
        table_4102[50] = v4854_4153;
        table_4102[51] = v4855_4154;
        ifv_4156 = table_4102[ifv_4099];
    }
    else
    {
        ifv_4156 = (w32) 5775200;
    }
    app_4160 = p4763 - ifv_4156;
    return_4097 = app_4160;
    return return_4097;
}
```

Listing F.31: C implementation of actuation_unit_impl.

```c
#include "common.h"
#include "actuation_logic.h"
#include "../generated/C/actuation_unit_impl.c"
```

Listing F.32: C implementation of actuator_generated_C.

```c
#define ActuateActuator ActuateActuator_generated_C
#define actuate_devices actuate_devices_generated_C
#include "../components/actuator.c"
#include "../generated/C/actuator_impl.c"
```

Listing F.33: C implementation of saturation_generated_C.

```c
#include "common.h"
#include "../generated/C/saturation_impl.c"
```

Listing F.34: C implementation of actuation_unit_generated_SystemVerilog.

```c
#ifdef PLATFORM_HOST
#include "../generated/SystemVerilog/verilator/actuate_d0/VActuate_D0.h"
#include "../generated/SystemVerilog/verilator/actuate_d1/VActuate_D1.h"
#include <stdio.h>
#else
#include "printf.h"
#endif

#define Actuate_D0 Actuate_D0_generated_SystemVerilog
#define Actuate_D1 Actuate_D1_generated_SystemVerilog
#define actuation_unit_step actuation_unit_step_generated_SystemVerilog
#include "../components/actuation_unit.c"

#ifdef PLATFORM_HOST
static VActuate_D0 actuate_d0;
static VActuate_D1 actuate_d1;
uint8_t Actuate_D0(uint8_t trips[3][4], uint8_t old) {
    actuate_d0.old = old;
    for(int b = 0; b < 12; ++b) {
        memcpy((uint8_t *)actuate_d0.trips + b, (uint8_t *)trips + (11 - b), 1);
    }
    actuate_d0.eval();
    DEBUG_PRINTF(("<actuation_unit_generated_SystemVerilog.c> actuate_base: device=0x0, old=0
        ↪ x%X, out=0x%X,trips=[",
            old, actuate_d0.out));
    for (int i = 0; i < 3; ++i) {
        DEBUG_PRINTF(("["));
        for (int div = 0; div < 4; ++div) {
        DEBUG_PRINTF(("%u,",trips[i][div]));
        }
        DEBUG_PRINTF(("],"));
    }
    DEBUG_PRINTF(("]\n"));
    return actuate_d0.out;
}

uint8_t Actuate_D1(uint8_t trips[3][4], uint8_t old) {
    actuate_d1.old = old;
    for(int b = 0; b < 12; ++b) {
        memcpy((uint8_t *)actuate_d1.trips + b, (uint8_t *)trips + (11 - b), 1);
    }
    actuate_d1.eval();
    DEBUG_PRINTF(("<actuation_unit_generated_SystemVerilog.c> actuate_base: device=0x1, old=0
        ↪ x%X, out=0x%X,trips=[",
            old, actuate_d1.out));
```

```
    for (int i = 0; i < 3; ++i) {
        DEBUG_PRINTF(("["));
        for (int div = 0; div < 4; ++div) {
        DEBUG_PRINTF(("%u,",trips[i][div]));
        }
        DEBUG_PRINTF(("],"));
    }
    DEBUG_PRINTF(("]\n"));
    return actuate_d1.out;
}
#else
#include "bsp.h"
#include "platform.h"

uint8_t actuate_base(uint8_t trips[3][4], uint8_t old, uint8_t id);

uint8_t actuate_base(uint8_t trips[3][4], uint8_t old, uint8_t id)
{
    // NOTE: reverse ordering: 2->1->0 goes to 0->1->2
    // Set value for trip value 0
    size_t idx = 0;
    write_reg(ACTUATION_REG_GENERATED_TRIP_2, (uint32_t) (trips[idx][3] << 24| trips[idx][2]
        ↪ << 16 | trips[idx][1] << 8 | trips[idx][0]));
    idx++;
    write_reg(ACTUATION_REG_GENERATED_TRIP_1, (uint32_t) (trips[idx][3] << 24| trips[idx][2]
        ↪ << 16 | trips[idx][1] << 8 | trips[idx][0]));
    idx++;
    write_reg(ACTUATION_REG_GENERATED_TRIP_0,(uint32_t) (trips[idx][3] << 24| trips[idx][2]
        ↪ << 16 | trips[idx][1] << 8 | trips[idx][0]));

    // trigger the actuation
    // wdata[0] - value of `old` argument
    // wdata[1] - which actuator to actuate
    write_reg(ACTUATION_REG_GENERATED_BASE, (uint32_t)( id << 1 | old));

    // Get actuation results (only the last bit is pertinent for True/false)
    uint8_t res = (uint8_t) (read_reg(ACTUATION_REG_GENERATED_RESULT) & 0x1);

    DEBUG_PRINTF(("<actuation_unit_generated_SystemVerilog.c> actuate_base: device=0x%X, old
        ↪ =0x%X, out=0x%X,trips=[", id, old, res));
    for (int i = 0; i < 3; ++i) {
        DEBUG_PRINTF(("["));
        for (int div = 0; div < 4; ++div) {
        DEBUG_PRINTF(("%u,",trips[i][div]));
        }
        DEBUG_PRINTF(("],"));
    }
    DEBUG_PRINTF(("]\n"));
    return res;
}

uint8_t Actuate_D0(uint8_t trips[3][4], uint8_t old)
{
    return actuate_base(trips, old, 0);
}

uint8_t Actuate_D1(uint8_t trips[3][4], uint8_t old)
{
    return actuate_base(trips, old, 1);
}
#endif
```