RDE Example

High-Assurance Cyber-Physical Systems

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)
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

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)

(based on the requirements of IEEE Std 603-2018):

Queries	Level	Artifacts
What must it do?	Requirements	Lando, FRET
How do the parts interact?	Architecture	SysML
How will it do it?	Specifications	Cryptol
Is it built right?	Verification and correctness	Cryptol
What does it do?	Behavior	Verilog
Does it do the right thing?	Validation and testing	—
How do we build it?	Implementation	FPGA

Requirements

The RTS demonstrator treated requirements as formal, analyzable artifacts.

- **Lando** was used to rewrite natural language requirements into a **structured, semantic format**.
  - Enabled clarification, modularity, and alignment with domain concepts.
  - Served as a **bridge** between informal stakeholder intent and formal tools.
- **FRET (Formal Requirements Elicitation Tool)** captured precise, logic-based properties:
  - Expressed **assumptions** and **guarantees** over time and behavior.
  - Requirements were made **machine-checkable**, and suitable for **formal verification**.
  - Used as input to tools like **Cryptol** and **Frama-C**.

Together, Lando and FRET formed a pipeline from informal intent to formal verification.

Architecture

The RTS system architecture was modeled to support structure, safety, and traceability.

- The architecture captured the **decomposition of subsystems**:
  - Sensor logic, redundant voting, and actuation behavior
  - Clear separation of hardware and software responsibilities
- **SysMLv2** was used to express:
  - System components, interfaces, and data flows
  - Allocation of functions to hardware/software
  - Behavioral constraints and stakeholder interactions
- SysML enabled:
  - **Model-based traceability** across layers
  - Planning for **verification and validation**
  - Mapping from requirements to implementation

Architecture modeling provided a foundation for structured refinement and assurance.

Specifications

Specifications define **how the RTS system will meet its requirements** — describing intended behavior, structure, and responses.

In the HARDENS project, specifications were developed using:

- **Architecture-informed logic** to define how sensor inputs are processed, compared, and acted upon
- Precise rules for **redundant voting**, **fault masking**, and **actuator triggering**
- Conditions for **normal operation**, **error handling**, and **edge-case behaviors**
- Modeled behavior that accounts for timing, sequencing, and state transitions

These specifications were written in a form suitable for **formal verification**, and directly informed the development of **verifiable implementations** and **testable simulations**.

Verification and Correctness

Verification was a core focus of the RTS demonstrator:

- **Cryptol** was used as a formal verification tool:
  - Modeled sensor logic, redundancy, and actuation behavior
  - Theorems were proved against FRET specifications
  - Properties verified using **SAW** (Software Analysis Workbench)
- **Frama-C + ACSL** verified C implementations against behavior and safety contracts
- Hardware logic (SystemVerilog, Bluespec) was aligned and checked

This multi-layer verification ensured correctness across software and hardware.

Behavior

System behavior was expressed and simulated through:

- Temporal and logical properties in **FRET**
- Executable functional models in **Cryptol**
- Control logic verified in C and SystemVerilog

Behavioral properties were validated against expectations from the original requirements, ensuring correct responses to all relevant inputs.

Validation and Testing

Validation strategies included:

- **Simulation of Cryptol models** to explore behavior
- **Test benches** generated from specifications
- **Runtime monitors** aligned with formal properties
- Hardware validation using assertion-based test environments

These efforts ensured that observed behavior aligned with the modeled expectations and safety goals.

Implementation

Final implementation artifacts were grounded in the verified specifications:

- **Software**:
  - C code verified using Frama-C and SAW
  - Aligned with verified Cryptol specifications
- **Hardware**:
  - HDL implementations cross-checked with functional models
  - Assertion coverage supported traceability

All implementation artifacts were packaged with evidence of correctness, traceability to requirements, and structured documentation for review.

Summary: The RTS Engineering Stack (Revised)

Layer	Approach Used in HARDENS RTS
Requirements	Natural language refined with Lando and FRET
Architecture	Modeled component breakdown and control flow
Specifications	Defined in FRET as formal, analyzable contracts
Verification/Correctness	Cryptol + SAW, Frama-C, assertion checking
Behavior	Expressed in FRET and executable in Cryptol
Validation & Testing	Simulated models, test benches, monitors
Implementation	Verified C and HDL with evidence

This workflow enabled end-to-end assurance through formal structure and layered verification.