Simulink Design Verifier 2.0

Product Presentation

Denizhan Alparslan, Ph.D.
Agenda

- Introduction: Design Verification Challenge
  - Discover unanticipated functionality
- Part 1: Identifying Design Errors Early
- Part 2: Verifying Design Against Requirements
  - Model and validate requirements using models
  - Prove design correctness
- Part 3: Model Coverage Analysis
  - Generate test vectors
  - Measure model coverage
- Part 4: What’s New in Simulink Design Verifier
Design Verification Challenge
Discover Unanticipated Functionality

- Test for unanticipated (unwanted) functionality
  - Example: Thrust reversers shall not (never, by design) deploy during flight

- Help:
  - Process: Industry standards such as DO-178B, ISO 26262
  - Rigor: Systematic testing (conditions, decisions, MC/DC)
  - Math: Formal methods
How can Simulink Design Verifier help?

- Model and analyze functional requirements
- Identify design errors
- Generate tests and analyze model coverage
- Verify generated code in cosimulation
Early Design Verification with Simulink Design Verifier

**Early design verification**
- Requirements-based analysis
- Design error detection
- Property proving

**Later – design done**
- Test coverage analysis
- Code verification (SIL/PIL)
TRW Automotive Develops and Tests Electric Parking Brake Using Simulink and Simulink Design Verifier

**Challenge**
Design tests for an electric parking brake control system

**Solution**
Use Simulink Design Verifier to automatically generate tests that maximize model coverage and enable systematic design verification

**Results**
- Test development time reduced from days to hours
- 100% model coverage achieved
- Formal testing begun two months into the project

“Everyone knows that errors are much less expensive to fix when you find them early. With Simulink Design Verifier, we build on the advantages of Model-Based Design by performing formal testing in the first phases of development.”

Christoph Hellwig
TRW

Link to user story
Simulink Design Verifier 2.0

Key Features

- Polyspace and Prover Plugin formal analysis engines
- Detection of dead logic, integer and fixed-point overflows, division by zero, and violations of design properties
- Blocks and functions for modeling functional and safety requirements
- Test vector generation from functional requirements and model coverage objectives, including condition, decision, modified condition/decision (MC/DC), and signal range
- Property proving, with generation of violation examples for analysis and debugging
- Fixed-point and floating-point model support
Identifying Design Errors Early
Identifying Design Errors Early

Automatic identification of hard-to-find design inconsistencies in the model without running simulation

- Integer overflow
- Division by zero
- Dead logic
- Assertion violation
Example Design Error Found

Dead Logic

- Certain designed functionality can *never* be activated.

- Typical implications:
  - Design can’t meet requirements.
  - Design generates dead code.
Example Design Error Found
Division by Zero, Overflow

- Certain valid input data can cause non-deterministic behavior or exceptions.
- Typical implication:
  - Incomplete or incorrect specification
Example Design Error Found
Assertion Violation

- Assertions are blocks you can add to your design to:
  - Detect faulty behavior
  - Monitor design and generated code running in simulation
- Simulink Design Verifier can provide you with the test cases that can trigger assertions
Verifying Design Against Requirements
Working with Formal Requirements

Design

Requirements

Verification

Properties/test specification

Verification Model
(references design model)

Design Verifier

Results

C

Models for code generation do not contain Simulink Design Verifier blocks

V1.0

V2.0

V3.0

Simulink Design Verifier Threat Reverser

design deploy

sensorData

asWarn

tWarn

wowWarn

wssWarn

Design Model

Safety Properties

design deploy

sensorData

asWarn

tWarn

wowWarn

wssWarn

Pattern library
Formalizing Requirements Into Properties

Model Requirements → Prove Design Correctness

Must do → Must never do

Chapter 3. Proof Objectives Status

Table of Contents
Objectives Proven Valid

Objectives Proven Valid

<table>
<thead>
<tr>
<th>#</th>
<th>Type</th>
<th>Model Item</th>
<th>Description</th>
<th>Counterexample</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Proof objective</td>
<td>Requirements Model/REQ 1.2.1 Airspeed/Verification Point: No Deploy when in Flight Airspeed</td>
<td>Objective: T</td>
<td>n/a</td>
</tr>
<tr>
<td>2</td>
<td>Proof objective</td>
<td>Requirements Model/REQ 1.2.2 Wheelspeed/Wheelspeed Valid</td>
<td>Objective: T</td>
<td>n/a</td>
</tr>
</tbody>
</table>
Examples of Formalized Requirements

Explicit descriptions of required behavior – functional or safety requirement

- **Primitive:**
  - Objectives, proofs

- **Invariant:**
  - $A > B \Rightarrow C=0$

- **Temporal:**
  - $A > B$ for 10 time steps $\Rightarrow C=0$ within 5 time steps

- **Complex, state-based**
  - Stateflow, MATLAB functions

Example invariant:
- Always identical, for every time step, including initialization, all modes of operation
Functional Requirements
Must Do (Test Case)

Must demonstrate opening and closing the valve when [Condition]
Safety Requirements
Must Never Do (Proof)

Thrust reverser shall not deploy when [Condition]
Validation of Formal Verification Results
Simulation / Debugging

Requirement models (properties) cosimulate with the design. Simulation driven by counterexamples.
Validation of Property Proving Results
Simulation / Model Coverage

Model coverage of Simulink Design Verifier objectives
Model Coverage Analysis
Model Coverage Analysis

Generate Tests

Measure Coverage in Simulation

Formal Methods

Simulation
Verify Generated Code

1. Run requirements-based tests
2. Generate test vectors for missing model coverage objectives
3. Review generated test vectors
4. Update requirements-based tests
5. Execute tests on the generated code in SIL and PIL
6. Compare results
Using Test Vectors for Code Verification

A. Exporting test data into code testing tools

B. Cosimulating via S-function wrappers
Applying Formal Methods

Technology Limitations

- *Simulink Design Verifier supports discrete time systems only*

- Use of non-linear functions and long timers may require abstraction, time scaling, or other additional strategies.
  - In some cases, theorem prover requires additional information to solve the problem.

- Proof of property provides levels of confidence beyond test generation and violation detection, but it is also harder to achieve. Complete proof may require optimization of the design for the purpose of verification.
What’s New in Simulink Design Verifier
Extending Simulink Block Support

- Dedicated analysis engine for nonlinear arithmetic and math operations
- Support for Stateflow absolute-time temporal logic operators
- Model reference
  - Automatic stubbing for unsupported operations
  - Subsystem replacement
- Embedded MATLAB Subset Support
  - Stateflow truth tables
- Simulink and Stateflow
  - Virtual buses
  - Block replacement
- Support for dead zone, dead zone dynamic, lookup table dynamic, probe (partial), and width
- Support for enumerated signals and parameters, additional Simulink blocks
- Simulink bus signals and bus objects support
- Fixed-point support
Making Definition of Verification Objectives Easier

Library of temporal operators (demo)

New Implies block
New property-proving examples and demos

Test condition, test objective
Proof assumption, proof objective
Verification subsystem
Stateflow functions for verification objectives and constraints

New temporal operator blocks: Detector, Extender, Within Implies

New Embedded MATLAB functions for verification objectives and constraints

2007a
2007b
2008a
2008b
2009a
2009b
2010a
2010b
2011a
TÜV Certification of Simulink Design Verifier

- TÜV SÜD certified:
  - Embedded Coder
  - Simulink Design Verifier
  - Simulink Verification and Validation
  - Polyspace products for C/C++
- For use in development processes which need to comply with IEC 61508, ISO 26262, or EN 50128

Note: The products listed above were not developed using certified processes.

MathWorks announcements:
www.mathworks.com/company/pressroom/articles/article17790.html (Initial certification)
www.mathworks.com/company/pressroom/articles/article39270.html (Recertification, ISO 26262 support)

TÜV SÜD certificate database:
Conclusion

- Simulink Design Verifier can automatically discover the following types of design errors:
  - Division by zero
  - Integer overflow
  - Dead logic
  - Assertion violations

- Definition of functional test objectives and design properties using the supplied operator blocks enable formal requirements modeling and verification