MATLAB EXPO 2015
KOREA
2015년 5월 21일 목요일
인터컨티넨탈 코엑스, 서울
Software Design and Verification for Model and Code

Presenter: Sang-Ho Yoon
Senior Application Engineer
Today’s Agenda

- Introduction to V&V in Model-Based Design
- Verification Activities in MBD
- Model Verification
  - Design Review
  - Static Analysis
  - Dynamic Test
- Code Generation and Verification
  - Code Review
  - Static Analysis
  - Equivalence Test
- Questions and Answers
Introduction to V&V in Model-Based Design
Development Process

Requirements

Ind. Standards

System-Level Specification

Environment

Physical Components

Algorithms

Subsystem Design

C, C++

MCU

DSP

HDL

FPGA

ASIC

Verification and Validation

Code Verification and Validation

Integration Testing

User Acceptance Testing

Complete Integration & Test

System-Level Integration & Test

Subsystem Integration & Test

Generate

Subsystem Implementation
Price of Late Verification and Validation
Relative Cost to Fix Defects

- High cost of fixing defects detected late
- Difficult to know when “enough” testing is done
- As design complexity increases, time and cost to test does too

Reference Workflow for Verification and Validation in Model-Based Design

Model Verification

1. Textual requirements
2. Executable specification
3. Module and integration testing at the model level
4. Review and static analysis at the model level

Code Verification

1. Generated C code
2. Object code
3. Equivalence testing
4. Prevention of unintended functionality

Integration of Test Case Generation into Model-Based Design Process
Verification Activities in MBD
SOFTWARE VERIFICATION PROCESS

This section discusses the objectives and activities of the software verification process. Verification is a technical assessment of the outputs of the software planning process, software development processes, and the software verification process. The software verification process is applied as defined by the software planning process (see 4) and the Software Verification Plan (see 11.3). See 4.6 for the verification of the outputs of the planning process.

Verification is not simply testing. Testing, in general, cannot show the absence of errors. As a result, the following sections use the term "verify" instead of "test" to discuss the software verification process activities, which are typically a combination of reviews, analyses, and tests.

*DO-178C Section 6.0
Verification Activities in MBD

- **Model Verification**
  - Design Review
    - Requirement Traceability
    - Report Generation
  - Static Analysis
    - Model Standards Checking
    - Design Error Detection
    - Prove Design Correctness
  - Dynamic Test
    - Simulation-based Functional Test
    - Coverage Analysis

- **Code Generation and Verification**
  - Code Review
    - Code Generation Report
  - Static Analysis
    - Code Metrics and Coding Rule Checking
    - Formal Verification (Abstract Interpretation)
  - Equivalence Test
    - SIL(Software-In-the-Loop) and PIL(Processor-In-the-Loop)
Model Verification
Design Review

- Requirement Traceability
- Report Generation

Static Analysis

- Modeling Standard Checking
- Design Error Detection
- Prove Design Correctness
- Model Slice

Dynamic Test

- Simulation-based Functional Test
- Coverage Analysis
Algorithm Model

- Traceability is required for functional safety standards

Display System Requirements

Component Modes

Landing Gear

The aircraft contains 2 landing gear units, one on the right and another on the left. The landing gear units can be locked when they are fully retracted (up) or fully extended (down). Both the landing gear units should work in tandem. In other words, they should both be locked at the end positions and un辽ked for extending/awakening together. Different outputs are generated when both the units are locked, only one or none is locked. The control logic, which is the graywine, should be displayed in this case.

Inputs

- The state of the landing gear will be contained in a bus object $\text{AirData}$. This bus object is defined as follows:

<table>
<thead>
<tr>
<th>Name</th>
<th>Data Type</th>
<th>Dimension</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pitch</td>
<td>double</td>
<td>1</td>
</tr>
<tr>
<td>Velocity</td>
<td>double</td>
<td>1</td>
</tr>
<tr>
<td>LandingStatus</td>
<td>double</td>
<td>1</td>
</tr>
<tr>
<td>Angle</td>
<td>double</td>
<td>1</td>
</tr>
<tr>
<td>Status</td>
<td>boolean</td>
<td>1</td>
</tr>
</tbody>
</table>

- The $\text{status}$ element of the bus will be used to determine the output of the system $\text{LandingGearMode}$.

Output

- The output of the system shall be an integer of type double, $\text{LandingGearMode}$.

Logic

- $\text{LandingStatus} = $ true shall indicate a locked status for left landing gear unit.
- $\text{LandingStatus} = $ false shall indicate an unlocked status for left landing gear unit.
- $\text{LandingStatus} = $ true shall indicate a locked status for right landing gear unit.
- $\text{LandingStatus} = $ false shall indicate an unlocked status for right landing gear unit.

Functional Requirements

Model
Requirements Traceability
Simulink Verification and Validation

- Traceability analysis of models relative to
  - System requirements
  - Design, interface descriptions
  - Change requests
  - Defect reports

- Standards and Certification
  - ISO 26262, IEC 61508, DO 178, EN 50128
  - Other industry standards (CMMI, SPICE, etc.)
Tracing Requirements ↔ Model
Simulink Verification and Validation

- Creating links between textual documents and model objects
Requirements Traceability – Report
Simulink Verification and Validation

- Requirements Report provides screenshots of the model and lists all the associated requirements.
Report for Model Review
Simulink Report Generator

- Design and generate reports for MBD
- Interactive design reviews
- Model Comparison and Merge
Design Review

- Requirement Traceability
- Report Generation

Static Analysis

- Modeling Standard Checking
- Design Error Detection
- Prove Design Correctness
- Model Slice

Dynamic Test

- Simulation-based Functional Test
- Coverage Analysis
Potential Error

- Is there a potential error in this model?
  - It depends…
The Need for Static Analysis

- How about now?

When generating code:
- Floating-point precision issues may lead to incorrect comparison results

- Ports do not follow established naming conventions

Is this a production model?
- Implementation requires a fixed-step, discrete solver
Design Review
• Requirement Traceability
• Report Generation

Static Analysis
• Modeling Standard Checking
• Design Error Detection
• Prove Design Correctness
• Model Slice

Dynamic Test
• Simulation-based Functional Test
• Coverage Analysis
Modeling Guidelines

- **MAAB Style Guides**
  - First version released in April, 2001
  - Collaboration by industry leaders in US, Japan, Europe: GM, Ford, Chrysler, Toyota, Daimler, John Deere, Delphi, Ricardo and others

- **Modeling Guidelines for High-Integrity Systems**
  - Leverage industry-best practices and MathWorks tool expertise when developing high-integrity systems
  - ISO 26262, IEC 61508, DO-178B/C, and MISRA-C
Modeling Standards Checking in Simulink
Simulink Verification and Validation

- Analysis Engine
  - Model Advisor (Simulink)

- Checks
  - Readability and Semantics
  - Performance and Efficiency
  - Change and Configuration Management
  - Potential runtime errors
Design Review
- Requirement Traceability
- Report Generation

Static Analysis
- Modeling Standard Checking
- Design Error Detection
- Prove Design Correctness
- Model Slice

Dynamic Test
- Simulation-based Functional Test
- Coverage Analysis
Design Error Detection with Formal Methods
Simulink Design Verifier

Detect hard-to-find design errors before simulation

- Dead logic
- Division by zero
- Range violation
- Integer overflow
- Assertion violation
- Out of bound array access
Design Review
- Requirement Traceability
- Report Generation

Static Analysis
- Modeling Standard Checking
- Design Error Detection
- Prove Design Correctness
- Model Slice

Dynamic Test
- Simulation-based Functional Test
- Coverage Analysis
Verifying Design Against Requirements
Simulink Design Verifier

1. Formal model (System)
2. Formal Property
3. Improve algorithm, or requirements

Algorithm (System)

Functional and Non-Functional Requirement

Formal Methods Engine

Informal

Formal

MATLAB EXPO 2015
Example of Properties

- Simple True / False
  - Simple logic blocks
  - Truth table

- Temporal logic
  - Temporal Logic blocks (part of SLDV demo)
    - Example of using basic templates as building blocks for complex property.
  - Imply blocks (part of SLDV blockset)
    - \(~A\) or \(B\)
  - Stateflow / Embedded MATLAB Functions
Requirements Proving

Design model + Specified properties

Formal Methods

Proof

OR

Counterexample
- Requirement Traceability
- Report Generation

### Static Analysis

- Modeling Standard Checking
- Design Error Detection
- Prove Design Correctness
- Model Slice

### Dynamic Test

- Simulation-based Functional Test
- Coverage Analysis
Model Slicer: Test and Debug Complex Models
Simulink Design Verifier

Can you show me just what I really care about?

MATLAB EXPO 2015
Model Slicer Highlighting

- Propagates signals and data of interest through your design:
  - Forward
  - Backward
  - Bidirectional

- Shows effect of a simulation time window

- Supports interactive selecting or excluding blocks/signals
Creating Model Slices

I found the important part of my model, now how can I isolate that for simulation, debugging, etc.?

Model Slices
Reduce your original design model to just the necessary and sufficient parts for the area of interest.
Slice Model Generation

- Eliminates:
  - Unneeded blocks
  - Unused ports
  - Unnecessary signals
- Simplifies hierarchy and signal flow

Resulting Models
- Easier to understand
- Simpler to analyze
Design Review

- Requirement Traceability
- Report Generation

Static Analysis

- Modeling Standard Checking
- Design Error Detection
- Prove Design Correctness
- Model Slice

Dynamic Test

- Simulation-based Functional Test
- Coverage Analysis
Module Test-Harness

Test Cases (Signal Builder)

Model (Model Block)

Output Check (Assertion Block)
Test Cases to Signal Builder

1. TC01 – Passing Maneuver
   This test case is tested with 90% throttle value to check passing maneuver and steady gear number increase.

   ![Graph showing speed and throttle with passing maneuver]

2. TC02 – Gradual Acceleration
   This test case tests gear change behavior with gradual speed increase with steady increase of throttle pedal position.

   ![Graph showing speed and throttle with gradual acceleration]

---

**Test Cases**

MATLAB EXPO 2015
Model Coverage Report

Simulink Verification and Validation

Coverage metrics identifies untested portions of your model

Coverage Report for sbr

Tests

Test 1

Started Execution: 27-Feb-2008 13:36:21
Ended Execution: 27-Feb-2008 13:36:21

Summary

<table>
<thead>
<tr>
<th>Model Hierarchy/Complexity</th>
<th>D1</th>
<th>C1</th>
<th>Test 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. sbr</td>
<td>100%</td>
<td></td>
<td>100%</td>
</tr>
<tr>
<td>2. Input</td>
<td>3 56%</td>
<td></td>
<td>NA</td>
</tr>
<tr>
<td>3. Antomatic Tests</td>
<td>2 50%</td>
<td></td>
<td>NA</td>
</tr>
<tr>
<td>4. Output Assertive</td>
<td>2 100%</td>
<td></td>
<td>100%</td>
</tr>
<tr>
<td>5. Verification Subsystem</td>
<td>2 100%</td>
<td></td>
<td>100%</td>
</tr>
<tr>
<td>6. SBR Logic</td>
<td>2 74%</td>
<td></td>
<td>NA</td>
</tr>
<tr>
<td>7. SBR</td>
<td>2 74%</td>
<td></td>
<td>NA</td>
</tr>
<tr>
<td>8. SFB SFB</td>
<td>2 74%</td>
<td></td>
<td>NA</td>
</tr>
<tr>
<td>9. SFB KEY ON</td>
<td>15 71%</td>
<td></td>
<td>NA</td>
</tr>
<tr>
<td>10. SF SB UNFASTEN</td>
<td>8 75%</td>
<td></td>
<td>NA</td>
</tr>
<tr>
<td>11. SFB HIGH SPEED</td>
<td>4 75%</td>
<td></td>
<td>NA</td>
</tr>
<tr>
<td>12. SBR Logic1</td>
<td>5 5%</td>
<td></td>
<td>NA</td>
</tr>
<tr>
<td>13. SBR</td>
<td>5 5%</td>
<td></td>
<td>NA</td>
</tr>
</tbody>
</table>
Improving Test Suite
Simulink Design Verifier

- Generating tests to reach coverage criteria

Test generation from model

Test inputs that ensure complete coverage
Relational Boundary Coverage (R2014b)

- Verify testing around boundary points (Part of DO-178C)

Integer Relation

Floating-point Relation

<table>
<thead>
<tr>
<th>Relation</th>
<th>Lhs just &lt; Rhs</th>
<th>Lhs just &gt; Rhs</th>
</tr>
</thead>
<tbody>
<tr>
<td>throt - max_throt</td>
<td>0%</td>
<td></td>
</tr>
<tr>
<td>[-tol..0]</td>
<td>0/0</td>
<td></td>
</tr>
<tr>
<td>(0..tol]</td>
<td>0/0</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Relation</th>
<th>Lhs just &lt; Rhs</th>
<th>Lhs just &gt; Rhs</th>
</tr>
</thead>
<tbody>
<tr>
<td>throt - min_throt</td>
<td>0%</td>
<td></td>
</tr>
<tr>
<td>[-tol..0]</td>
<td>0/0</td>
<td></td>
</tr>
<tr>
<td>(0..tol]</td>
<td>0/0</td>
<td></td>
</tr>
</tbody>
</table>

MATLAB EXPO 2015
How about Legacy Code?

- Use of Legacy Code Tool for introduction of existing C code on Simulink models

External C Function

MATLAB code specification
Coverage for C-code S-Functions

S-Function Instances
Included instances: slexSFcnSLDExample/Legacy code S-Function

Tests
Test 1

Summary

<table>
<thead>
<tr>
<th>File Contents</th>
<th>D1</th>
<th>C1</th>
<th>MCDC</th>
</tr>
</thead>
<tbody>
<tr>
<td>.categorize.c</td>
<td>100%</td>
<td>100%</td>
<td>100%</td>
</tr>
<tr>
<td>... categorize_input</td>
<td>100%</td>
<td>100%</td>
<td>100%</td>
</tr>
</tbody>
</table>

Details
1. File categorize.c

Functions: categorize_input (line 3)

Design

Coverage

C code

MATLAB EXPO 2015
Code Generation and Verification
Code Review

- Code Generation Report

Static Analysis

- Code Metrics and Coding Rule Checking
- Formal Verification (Abstract Interpretation)

Equivalence Test

- SIL(Software-In-the-Loop)
- PIL(Processor-In-the-Loop)
Code Generation Report

Code Generation Report for 'iir_ert'

Summary

Code generation for model "iir_ert"

<table>
<thead>
<tr>
<th>Model version</th>
<th>1.15</th>
</tr>
</thead>
<tbody>
<tr>
<td>Simulink Coder version</td>
<td>8.8 (R2015a) 09-Feb-2015</td>
</tr>
<tr>
<td>Source code generated on</td>
<td>Thu May 07 13:20:58 2015</td>
</tr>
</tbody>
</table>

Configuration settings at the time of code generation: click to open
Code generation objectives: Unspecified
Validation result: Not run
Traceability between Model and Code

- Hyperlink within C code files.
- Hyperlink back to model.
- Hyperlink from model to code.
Traceability Report

Traceable Simulink Blocks / Stateflow Objects / MATLAB Functions

Root system: iir_ert

Object Name | Code Location
---|---
<Root>/u | iir_ert.c:34

Subsystem: iir_ert/DocBlock

No traceable objects in this Subsystem.

Subsystem: iir_ert/IIR_filter

Object Name | Code Location
---|---
<S2>/Sum | iir_ert.c:31
<S2>/Unit Delay | iir_ert.c:20, 35, 39
<S2>/a1 | iir_ert.c:32
<S2>/a0 | iir_ert.c:33

Subsystem: iir_ert/Model Info

No traceable objects in this Subsystem.
Static Code Metrics Report

- Integrated into generated Simulink Coder Report
- Target independent analysis
- Includes
  - Number of files
  - Number of lines
  - Estimation of global RAM
  - Estimation of stack size
Code Review

- Code Generation Report

Static Analysis

- Code Metrics and Coding Rule Checking
- Formal Verification (Abstract Interpretation)

Equivalence Test

- SIL (Software-In-the-Loop)
- PIL (Processor-In-the-Loop)
CATEGORY OF STATIC ANALYSIS OF USING TOOLS

- Error Prevention
- Error Detection
- Compiler Warnings
- Code Metrics, and Coding Rules
- Bug Findings
  (False negative)
- Formal Methods
  (No False negative)
Polyspace PRODUCTS

- Polyspace Bug Finder
- Polyspace Code Prover

Error Prevention

Compiler Warnings

Code Metrics, and Coding Rules

Polyspace Bug Finder

Polyspace Code Prover

Error Detection

Bug Findings

Formal Methods

(MathWorks®)

MATLAB EXPO 2015
Software metrics produced by Polyspace

**Files**
- Lines
- Lines of code
- Comment density
- Estimated function coupling

**Function**
- Lines within body
- Executable lines
- Cyclomatic complexity
- Language scope
- Paths
- Calling functions
- Called functions
- Call occurrences
- Instructions
- Call levels
- Function parameters
- Goto statements
- Return points

**Project**
- Files
- Header files
- Recursions
- Protected shared variables
- Non-protected shared variables

**Software quality objectives**
- Define custom levels with thresholds to measure achievement of a quality level
- Support for HIS (Hersteller Initiative Software) metrics
Polyspace support for code rules compliance

- **MISRA C:2004**
  - 131 rules supported
  - 9 rules not statically enforceable
  - 2 rules not supported

- **MISRA AC AGC** -- application of MISRA-C:2004 for generated code
  - 88 obligatory rules are supported
  - 8 are not statically enforceable
  - 2 are not supported

- **MISRA C++:2008**
  - 185 of the 228 rules supported

- **JSF++:2005**
  - 157 of 234 rules supported

- **Customization**
  - Turn rules off / warning / error
  - Define custom naming conventions
  - Mark violations as reviewed or indicate future action

- **MISRA C:2012**
  - 6 Directives supported
  - 132 rules supported
Polyspace

Formal Methods based Static Code Analysis

- Exhaustively verify code
  - Detect and prove absence of runtime errors
  - Precisely determines and propagates variable ranges

- Languages supported
  - C, C++, and Ada

- Verify SW robustness
  - Analyze for full range operating conditions
  - Specified ranges of parameters and inputs

OR

Green: reliable safe pointer access

Red: faulty out of bounds error

Gray: dead unreachable code

Orange: unproven may be unsafe for some conditions

Purple: violation MISRA-C/C++ or JSF++ code rules

Range data tool tip
Fixing the Runtime Error in the Design
Trace and Fix Using PolySpace Model Link SL

\[ \text{Power} = \frac{\text{Voltage} \times \text{Voltage}}{1.74} + \text{Zi}; \]

/* Update for UnitDelay: '<Root>/Unit_Delay' */
\[ \text{Zi} = \text{Power}; \]

May Overflow
(establish traceability between PolySpace analysis and model)
Code Review

- Code Generation Report

Static Analysis

- Code Metrics and Coding Rule Checking
- Formal Verification (Abstract Interpretation)

Equivalence Test

- SIL(Software-In-the-Loop)
- PIL(Processor-In-the-Loop)
Software-in-the-Loop (SIL) Testing:
Verify Production Controller with Software-in-the-loop

Execution
- Host/Host
- Nonreal-time

Compiled C Code
S-Function (Windows DLL)
Processor-in-the-Loop Testing:
Verify Production Controller with Processor-in-the-loop

Execution
- Host/Target
- Nonreal-time
Q&A