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
- Generate

- Verification and Validation

- Subsystem Implementation
  - C, C++
  - HDL
  - MCU
  - DSP
  - FPGA
  - ASIC

- Complete Integration & Test
  - User Acceptance Testing
  - Integration testing
  - System-Level Integration & Test
  - Code Verification and Validation

- Development Process
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

- Module and integration testing at the model level
- Review and static analysis at the model level

Textual requirements → Executable specification → Model used for production code generation

Code Verification

- Equivalence testing
- Prevention of unintended functionality

Generated C code → Object code

Code generation → Compilation and linking
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

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 unlocked for extending/retreating together. Different outputs are generated whenever both the units are locked, when both are unlocked and when only one is locked. The output controls which warning message should be displayed to the user.

Inputs

- The state of the landing gear will be contained in a bus object: "InputData." 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>Engaged</td>
<td>double</td>
<td>1</td>
</tr>
<tr>
<td>Calibrated</td>
<td>double</td>
<td>1</td>
</tr>
<tr>
<td>ThrottlePosition</td>
<td>double</td>
<td>1</td>
</tr>
<tr>
<td>L1Engaged</td>
<td>double</td>
<td>1</td>
</tr>
<tr>
<td>R1Engaged</td>
<td>boolean</td>
<td>1</td>
</tr>
<tr>
<td>L1Locked</td>
<td>boolean</td>
<td>1</td>
</tr>
<tr>
<td>R1Locked</td>
<td>boolean</td>
<td>1</td>
</tr>
</tbody>
</table>

- The "Engaged" and "Locked" elements of the bus will be used to determine the output of the system: "OutputData."

Output

- The output of the system shall be an integer of type double: "OutputData".

Logic

- L1Engaged = true shall indicate an engaged state for left landing gear unit.
- R1Engaged = false shall indicate an unengaged state for right landing gear unit.
- L1Locked = true shall indicate a locked state for left landing gear unit.
- R1Locked = false shall indicate an unengaged state for right landing gear unit.

- L1Engaged = false shall indicate an unengaged state for left landing gear unit.
- R1Locked = false shall indicate an unengaged state for right landing gear unit.

- L1Engaged = true and R1Engaged = false shall indicate both landing gear units are engaged.

- L1Engaged = false and R1Engaged = true shall indicate both landing gear units are engaged.

- L1Engaged = true and R1Engaged = true shall indicate both landing gear units are engaged.

- L1Engaged = false and R1Engaged = false shall indicate both landing gear units are engaged.

- L1Locked = true and R1Locked = true shall indicate both landing gear units are locked.

- L1Locked = false and R1Locked = false shall indicate both landing gear units are locked.

- L1Locked = true and R1Locked = false shall indicate both landing gear units are locked.

- L1Locked = false and R1Locked = true shall indicate both landing gear units are locked.

- L1Locked = false and R1Locked = false shall indicate both landing gear units are locked.

- L1Locked = true and R1Locked = true shall indicate both landing gear units are locked.

- L1Locked = false and R1Locked = false shall indicate both landing gear units are locked.

- L1Locked = true and R1Locked = false shall indicate both landing gear units are locked.

- L1Locked = false and R1Locked = true shall indicate both landing gear units are locked.

- L1Locked = false and R1Locked = false shall indicate both landing gear units are locked.

- L1Locked = true and R1Locked = true shall indicate both landing gear units are locked.

- L1Locked = false and R1Locked = false shall indicate both landing gear units are locked.

- L1Locked = true and R1Locked = false shall indicate both landing gear units are locked.

- L1Locked = false and R1Locked = true shall indicate both landing gear units are locked.

- L1Locked = false and R1Locked = false shall indicate both landing gear units are locked.

- L1Locked = true and R1Locked = true shall indicate both landing gear units are locked.

- L1Locked = false and R1Locked = false shall indicate both landing gear units are locked.

- L1Locked = true and R1Locked = false shall indicate both landing gear units are locked.

- L1Locked = false and R1Locked = true shall indicate both landing gear units are locked.

- L1Locked = false and R1Locked = false shall indicate both landing gear units are locked.

- L1Locked = true and R1Locked = true shall indicate both landing gear units are locked.

- L1Locked = false and R1Locked = false shall indicate both landing gear units are locked.

- L1Locked = true and R1Locked = false shall indicate both landing gear units are locked.

- L1Locked = false and R1Locked = true shall indicate both landing gear units are locked.

- L1Locked = false and R1Locked = false shall indicate both landing gear units are locked.

- L1Locked = true and R1Locked = true shall indicate both landing gear units are locked.

- L1Locked = false and R1Locked = false shall indicate both landing gear units are locked.

- L1Locked = true and R1Locked = false shall indicate both landing gear units are locked.

- L1Locked = false and R1Locked = true shall indicate both landing gear units are locked.

- L1Locked = false and R1Locked = false shall indicate both landing gear units are locked.

- L1Locked = true and R1Locked = true shall indicate both landing gear units are locked.

- L1Locked = false and R1Locked = false shall indicate both landing gear units are locked.

- L1Locked = true and R1Locked = false shall indicate both landing gear units are locked.

- L1Locked = false and R1Locked = true shall indicate both landing gear units are locked.

- L1Locked = false and R1Locked = false shall indicate both landing gear units are locked.

- L1Locked = true and R1Locked = true shall indicate both landing gear units are locked.

- L1Locked = false and R1Locked = false shall indicate both landing gear units are locked.

- L1Locked = true and R1Locked = false shall indicate both landing gear units are locked.

- L1Locked = false and R1Locked = true shall indicate both landing gear units are locked.

- L1Locked = false and R1Locked = false shall indicate both landing gear units are locked.

- L1Locked = true and R1Locked = true shall indicate both landing gear units are locked.

- L1Locked = false and R1Locked = false shall indicate both landing gear units are locked.

- L1Locked = true and R1Locked = false shall indicate both landing gear units are locked.

- L1Locked = false and R1Locked = true shall indicate both landing gear units are locked.
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

Is this a production model?
- Implementation requires a fixed-step, discrete solver

- Ports do not follow established naming conventions
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

Model Advisor Interface
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

Informal

Functional and Non-Functional Requirement
Algorithm (System)

Formal Methods Engine
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
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
Model Slicer: Test and Debug Complex Models
Simulink Design Verifier

Design Model

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.

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

Model Hierarchy/Complexity:

<table>
<thead>
<tr>
<th>Model</th>
<th>Test 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>sbr</td>
<td>100%</td>
</tr>
<tr>
<td>sbr.Logo</td>
<td>100%</td>
</tr>
<tr>
<td>sbr.KEY_ON.</td>
<td>100%</td>
</tr>
<tr>
<td>SF.SBR</td>
<td>100%</td>
</tr>
<tr>
<td>SF.KEY_ON</td>
<td>100%</td>
</tr>
<tr>
<td>SF.SB.UNFASTEN</td>
<td>100%</td>
</tr>
<tr>
<td>SF.HIGH_SPEED</td>
<td>100%</td>
</tr>
</tbody>
</table>

[Image and text related to the model coverage report and test results]
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>Relational Boundary Coverage</th>
<th>33%</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lhs just &lt; Rhs</td>
<td>101/101</td>
</tr>
<tr>
<td>Lhs == Rhs</td>
<td>0/101</td>
</tr>
<tr>
<td>Lhs just &gt; Rhs</td>
<td>0/101</td>
</tr>
</tbody>
</table>

Lhs just < Rhs
Lhs just > Rhs

[throt > max_throt | throt < min_throt]/
Fail.INC

Relational Boundary

<table>
<thead>
<tr>
<th>throt - max_throt</th>
<th>0%</th>
</tr>
</thead>
<tbody>
<tr>
<td>[-tol..0]</td>
<td>0/0</td>
</tr>
<tr>
<td>(0..tol]</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>throt - min_throt</th>
<th>0%</th>
</tr>
</thead>
<tbody>
<tr>
<td>[-tol..0]</td>
<td>0/0</td>
</tr>
<tr>
<td>(0..tol]</td>
<td></td>
</tr>
</tbody>
</table>
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

S-Function

C code

/* Legacy code example */
signed char result;

if(input > 0.0 && input < threshold) {
    result = 0;
} else if(input >= threshold) {
    result = 1;
} else if(input < -threshold) {
    result = -1;
} else {
    result = 0;
}
return result;
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"

Model version        1.15
Simulink Coder version 8.8 (R2015a) 09-Feb-2015
C source code generated on Thu May 07 13:20:58 2015

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 back to model.

Hyperlink from model to code.

Hyperlink within C code files.
### Traceability Report

#### Traceable Simulink Blocks / Stateflow Objects / MATLAB Functions

<table>
<thead>
<tr>
<th>Object Name</th>
<th>Code Location</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;Root&gt;/u</code></td>
<td><code>iir_ert.c:34</code></td>
</tr>
</tbody>
</table>

- **Subsystem:** `iir_ert/DocBlock`  
  No traceable objects in this Subsystem.

#### Generated Code

- **Main file:** `ert_main.c`
- **Model files:**
  - `iir_ert.c`
  - `iir_ert.h`
  - `iir_ert_private.h`
  - `iir_ert_types.h`
- **Utility files:**
  - `rtwtypes.h`

- **Subsystem:** `iir_ert/IIR_filter`
<table>
<thead>
<tr>
<th>Object Name</th>
<th>Code Location</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;S2&gt;/Sum</code></td>
<td><code>iir_ert.c:31</code></td>
</tr>
<tr>
<td><code>&lt;S2&gt;/Unit Delay</code></td>
<td><code>iir_ert.c:20, 35, 39</code></td>
</tr>
<tr>
<td><code>&lt;S2&gt;/a1</code></td>
<td><code>iir_ert.c:32</code></td>
</tr>
<tr>
<td><code>&lt;S2&gt;/b0</code></td>
<td><code>iir_ert.c:33</code></td>
</tr>
</tbody>
</table>

- **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)
CATEGOR Y OF STATIC ANALYSIS OF USING TOOLS

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

- Error Prevention
- Error Detection
- Compiler Warnings
- Code Metrics, and Coding Rules
- Polyspace Bug Finder
- Polyspace Code Prover

Bug Findings (False negative)
Formal Methods (No False negative)
# 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
    OR
    - Specified ranges of parameters and inputs

```c
static void pointer_arithmetic (void) {
    int array[100];
    int *p = array;
    int i;
    for (i = 0; i < 100; i++) {
        *p = 0;
        i++;
    }
    if (get_bus_status() > 0) {
        *p = 5;
    } else {
        i++;
    }
    i = get_bus_status();
    if (i >= 0) {
        (*p - i) = 10;
    }
}
```

- **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

MATLAB EXPO 2015
Fixing the Runtime Error in the Design
Trace and Fix Using PolySpace Model Link SL

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