MATLAB EXPO 2016
KOREA
4월 28일 (목)
등록 하기 matlabexpo.co.kr
Simulation-based Test Management and Automation

Sang-Ho Yoon
Senior Application Engineer
Today’s Agenda

- Verification Activities in MBD
- Simulation-Based Test
- Manage and Automate Simulation-Based Tests
- Equivalence Test between Model and Generated Code
- Questions and Answers
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.
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 Inspection
  - Static Analysis
    - Code Metrics and Coding Rule Checking
    - Formal Verification (Abstract Interpretation)
  - Equivalence Test
    - SIL (Software-In-the-Loop) and PIL (Processor)
Simulation-based Test
Example 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/retracting together. Different outputs are generated when both 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 represented by LeftLock and RightLock signals processed from sensor data. The signals are defined as follows:

<table>
<thead>
<tr>
<th>Name</th>
<th>Data Type</th>
<th>Dimension</th>
</tr>
</thead>
<tbody>
<tr>
<td>LeftLock</td>
<td>boolean</td>
<td>1</td>
</tr>
<tr>
<td>RightLock</td>
<td>boolean</td>
<td>1</td>
</tr>
</tbody>
</table>

- The LeftLock and RightLock signals will be used to determine the output of the system: LandingGearMode.

Output

- The output of the system shall be an integer of type double, LandingGearMode.

<table>
<thead>
<tr>
<th>Name</th>
<th>Data Type</th>
<th>Dimension</th>
</tr>
</thead>
<tbody>
<tr>
<td>LandingGearMode</td>
<td>double</td>
<td>1</td>
</tr>
</tbody>
</table>

Logic

- LeftLock == true shall indicate a locked state for left landing gear unit.
- LeftLock == false shall indicate an unlocked state for left landing gear unit.
- RightLock == true shall indicate a locked state for right landing gear unit.
- RightLock == false shall indicate an unlocked state for right landing gear unit.
- If LeftLock and RightLock are both locked, the value of LandingGearMode will be 1.
- If LeftLock or RightLock, but not both, are unlocked, the value of LandingGearMode will be 2.
- If neither LeftLock nor RightLock are locked, the value of LandingGearMode will be 3.
Module Test-Harness

Test Cases (Signal Builder)

Test Harness

Model (Model Block)

Output Check (Assertions)
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>Test 1</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>DO</td>
</tr>
<tr>
<td>1. sbr</td>
<td>100%</td>
</tr>
<tr>
<td>2. . . Inputs</td>
<td>3%</td>
</tr>
<tr>
<td>3. . . . Automatic Tests</td>
<td>2%</td>
</tr>
<tr>
<td>4. . . . Outputs Assertions</td>
<td>2%</td>
</tr>
<tr>
<td>5. . . . Verification Subsystem</td>
<td>100%</td>
</tr>
<tr>
<td>6. . . SB Logic</td>
<td>9%</td>
</tr>
<tr>
<td>7. . . . SB sbr</td>
<td>2%</td>
</tr>
<tr>
<td>8. . . . . SB KEY ON</td>
<td>10%</td>
</tr>
<tr>
<td>9. . . . . . SB UNFASTEN</td>
<td>10%</td>
</tr>
<tr>
<td>10. . . . . . SB High Speed</td>
<td>10%</td>
</tr>
<tr>
<td>11. . . . . . SB Low Speed</td>
<td>10%</td>
</tr>
<tr>
<td>12. . . . SB Logic</td>
<td>5%</td>
</tr>
<tr>
<td>13. . . . sbr</td>
<td>10%</td>
</tr>
</tbody>
</table>

- Metric
  - Cyclomatic Complexity
  - Decision (D)

- Coverage
  - (This object) 2/4 (50%) decision outcomes
  - (This function, declaration) 0/5 (0%) decision outcomes

- Decisions analysis
  - Substate executed: 100%
  - States: "HIGH_SPEED" 89.9%/10.1%
  - States: "LOW_SPEED" 59.8%/40.2%

- Substate activated when parent starts:
  - States: "HIGH_SPEED" 100%
  - States: "LOW_SPEED" 50/50
How about Legacy Code?

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

S-Function Instances
Included instances: slexSFcnSLDVExample/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)

/* 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;
Simulink Test: Test Harness

If only a subsystem need to be tested…
Do I need to make a new model for it?

If a subsystem is a library reference…
Do I need to make a new model for it?

Additional Subsystems

Do I need to make multiple test harnesses for multiple test objectives?

If only a subsystem need to be tested…
Do I need to make a new model for it?
Test Harnesses

- Enables unit testing without requiring new model
- Synchronized testing environment
- Supports testing library blocks
How to Make Test Scenarios
Simulink Test: Test Sequence Block

- Design temporal or logic-based test scenarios
- Works with MATLAB Action Language
- Create complex test inputs and assessments
- Trouble shooting aids
Test Sequence: "verify" statement

- Streams to Simulation Data Inspector
- Integrates with Simulink Test Manager pass/fail
How to Automate Various Tests
Simulink Test: Test Manager

- Create Test Cases
- Group into Test Suites and Test Files
- Execute individual or batch
- View summary and detailed results
- Archive, export, report
Test Manager Coverage Integration

Collect and aggregate coverage using Simulink Test Manager

- Coverage collected and aggregated automatically
- New support for coverage on Simulink Test harnesses
- View coverage summary from test results
- Embed full coverage reports into a test manager report
### Summary of Simulink Test

<table>
<thead>
<tr>
<th>1. Test Harnesses</th>
<th>2. Test Sequence Block</th>
<th>3. Test Manager</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Synchronized, simulatable test environment</td>
<td>• Inputs and assessments based on logical, temporal conditions</td>
<td>• Author, execute, manage test cases • Review, export, report</td>
</tr>
</tbody>
</table>

![Test Harness](image)

Main Model

Component under test

![Test Sequence](image)

![Test Manager](image)
Equivalence Test
between Model and Generated Code
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)
Code Coverage for Generated Code with SIL
Summary

- Model Verification includes Reviews, Analyses and Tests
- Model-Based Tests can be efficiently managed with Simulink Test
- Simulink Test provides following functionalities
  - Test Harnesses
  - Test Sequence
  - Test Manager
The MathWorks

Change the world by

Accelerating the pace
of discovery, innovation, development, and learning

in engineering and science
Q&A