Verification and Validation Solutions for High Integrity Systems

Tiffany Liang
Application Engineer
MathWorks
Recommended Workflow
Detecting errors early in the development cycle

Model Design • Simulation (SL/SF)
Model Comparison • Merge (Report Generator)
Report Generation (Report Generator)

Requirements

Executable Requirements Model

Implementation Model (Fixed-point)

Source Code

Object Code

Auto Code-Generation (EC)
Traceability Report (EC)

Configuration Management (Simulink Project)

Interface to Requirements Management Tool (SLVV)
Formal Verification (SLDV)
Model Guideline Check (SLVV)
Model Coverage (SLVV)
Test Case Generation (SLDV)
Design Error Detection (SLDV)

SIL/PIL Test (EC)
Code Coverage (EC)
Static Code Analysis (Polyspace)
MathWorks benefits
Early verification and Validation

• Able to form small V-loops
• Able to detect errors early in the development cycle

- Model ⇔ Code consistency allows for Simulink simulation results to be considered “truth”.
- Early model verification is possible due to the ability to investigate floating-point models.
- Large team development made easy through highly customizable tool chain.
- Errors in object code detected easily through synchronization between simulations and SILS/PILS.

- Requirements Linking (SLVV)
- Formal Verification (SLDV)
- Model Guideline Check (SLVV)
- Model Coverage Test (SLVV)
- Test Case Auto-Generation (SLDV)
- Report Auto-Generation (Report Generator)
- SILS/PILS (EC)
- Runtime Error Identification (Polyspace)
Examples of High Reliability Applications

Airbags
✓ Operational delay following impact

Electronic Parking Brake
✓ Unintended braking during operation

Antiskid Brakes
✓ Unintended asymmetrical braking

Vehicle-to-vehicle distance control
✓ Insufficient deceleration within required time

Sources:
- ISO 26262 - 250 CAN messages with 2500 individual signals
Example: Door Lock Control System

Door Lock Control

- Auto-lock when vehicle in motion
- Auto-unlock during emergencies
Our First Topic

- **Model Design • Simulation (SL/SF)**
- **Model Comparison • Merge (Report Generator)**
- **Report Generation (Report Generator)**

**Requirements**

**Executable Requirements Model**

**Implementation Model (Fixed-point)**

**Source Code**

**Object Code**

**Configuration Management (Simulink Project)**

- **Interface to Requirements Management Tool (SLVV)**
- **Model Guideline Check (SLVV)**
- **Model Coverage (SLVV)**
- **Test Case Generation (SLDV)**
- **Design Error Detection (SLDV)**
- **Auto Code Generation (EC)**
- **Traceability Report (EC)**
- **SIL/PIL Test (EC)**
- **Code Coverage (EC)**
- **Static Code Analysis (Polyspace)**
Door Lock Control Software Requirements

1. Task Rate Requirements

   REQ101 – The software shall execute as a 100ms task rate.

2. Initialization Requirements

   REQ201 – The software shall initialize controls in the Unlock state.

3. Diagnosis Requirements

   REQ301 – The software shall determine the lock state of each door based on the lock positions.
   - Lock position is under 1mm: Unlock state
   - Lock position is over 4mm: Lock state
   - Otherwise: Neutral state

   REQ302 – The software shall determine the overall vehicle lock state based on all door lock positions.
   - All doors in lock position: Lock state
   - All doors in unlock position: Unlock state

   REQ303 – The software shall determine the overall vehicle lock state to be in failure state due to lock failure in the case where there is no response to a door lock request in under 2 seconds.

4. Door Lock Request Requirements

   REQ401 – The doors shall automatically lock when the vehicle speed is above 5km/h for over 2 seconds and the engine is operating.

   REQ402 – The door locks shall automatically release after the airbags deploy.
Door Lock Model

Simulink / Stateflow

Increased Readability / Productivity through Graphical Modeling
Door Lock Test Model

Simulink / Simscape

Able to execute various tests using the control model

Model Block used to call control model

Test Input

Fail On/Off Switch

Plant Model

Simulation vs. Expected Results Comparison
Requirements & Logic Testing through Simulation

**Simulink / Stateflow**

- Early verification of entire system incl. plant behavior
- Investigation of failure/anomaly modes (difficult on H/W)

![Test data definition in Signal Builder](image1)

![Simulation Results](image2)
MATLAB/Simulink Products

**MATLAB**
- Easy data processing
- Concise programming language
- Abundant mathematical functions · file I/O
- 2-D/3-D visualization functionality

**Simulink**
- Block diagram modeling
- Abundant block library
- High-precision time simulation

**Stateflow**
- Flowcharts, State Diagrams, State Transition Tables

---

Technical Computing Environment

Model-Based Design Environment
Model Difference Comparisons

Simulink Report Generator

- Generate reports on difference comparisons between 2 models
  - Compatible with Simulink Project and version management software (i.e. Subversion)

Green: Component mismatch
Red: Parameter mismatch
The Next Topic

Requirements

Executable Requirements Model

Implementation Model (Fixed-point)

Source Code

Object Code

Configuration Management (Simulink Project)

Model Guideline Check (SLVV)
Model Coverage (SLVV)
Test Case Generation (SLDV)
Design Error Detection (SLDV)

Interface to Requirements Management Tool (SLVV)

Formal Verification (SLDV)

Model Design·Simulation (SL/SF)
Model Comparison·Merge (Report Generator)
Report Generation (Report Generator)

Auto Code-Generation (EC)
Traceability Report (EC)

SIL/PIL Test (EC)
Code Coverage (EC)
Static Code Analysis (Polyspace)
Ensure Traceability
Requirement ⇔ Model ⇔ Test
Simulink Verification & Validation

3. Diagnosis Requirements

REQ301 – The software shall determine the lock state of each door based on the lock positions.
  - Lock position is under 1mm: Unlock state
  - Lock position is over 4mm: Lock state
  - Otherwise: Neutral state

REQ302 – The software shall determine the overall vehicle lock state based on all door lock positions.
  - All doors in lock position: Lock state
  - All doors in unlock position: Unlock state

REQ303 – The software shall determine the overall vehicle lock state to be in failure state due to lock failure in the case where there is no response to a door lock request in under 2 seconds.
Model Coverage for Measuring Test Completeness Level

Simulink Verification & Validation

Check for insufficient testing

Cumulative coverage results on multiple tests

Identify areas of missing coverage
Generate Tests for Full Model Coverage

Simulink Design Verifier

- Automatic test generation
- Suitable for equality tests

※ Able to generate missing tests based on user-defined tests

Test Harness Model

Auto-generated Test Data
Identification of Software Design Errors

Simulink Design Verifier

- Check for risks of software design errors prior to implementation
  Integer overflow, division by zero, range violations, dead logic

Overflow Identified

No risk of overflow

Fix

Example: Modify block parameter
Model Verification & Validation Products

Simulink Verification and Validation™ (SLVnV)

Measure Model Coverage
- Model Coverage Report
  - Decision
  - Condition
  - MC/DC

Traceability
- Model to Requirement
  - Word
  - Excel
  - DOORS
  - MKS Integrity
- Requirement to Model
  - Word/Excel/DOORS/MKS Integrity

Model Checker (Model Advisor)
- GUI for Model Checks
- Automate corrections on warnings
- Report Generation
- Add Custom Checks

Test Data Sufficiency Check

Requirement Sufficiency Check

Automate Model Checking

Simulink Design Verifier™ (SLDV)

Design Error Detection
- Auto-detect design errors
  - Division by zero
  - Range overflow
  - Deadl Logic
  - Saturation overflow
  - Out of bounds access

Auto-Generate Test Cases
- Controller Model
  - Analysis
- 100% Coverage Test Data

Property Proving (Formal Methods)
- Controller Model
  - Reqmt Spec
- Verification Model
  - V&V Spec

Certify Correct Behavior
The Final Topic

Requirements

Executable Requirements Model

Implementation Model (Fixed-point)

Source Code

Object Code

Model Design · Simulation (SL/SF)
Model Comparison · Merge (Report Generator)
Report Generation (Report Generator)

Interface to Requirements Management Tool (SLVV)
Formal Verification (SLDV)

Model Guideline Check (SLVV)
Model Coverage (SLVV)
Test Case Generation (SLDV)
Design Error Detection (SLDV)

Auto Code-Generation (EC)
Traceability Report (EC)

Code Coverage (EC)
Static Code Analysis (Polyspace)

Configuration Management (Simulink Project)
Generate Code from Controller Model

Embedded Coder

- Auto-generate C-code of high readability/efficiency
- Option settings for variable attributes, function settings, code style, etc.
- Auto-generate scaling for fixed-point design

```c
if (LockMode == FAILURE) {
    LockRequest = FALSE;
} else {
    LockRequest = ((spd_time >= Speed_time) && Engine_ON && (!Airbag_ON));
}
```
Ensuring Traceability between Requirements, Models, and Code

**Embedded Coder / Simulink Report Generator**

- Reflect model specifications in generated code
- Distribute reports with model views (html)

**Code⇔Document Link**

**Code⇔Model Link**

**Code Generation Report**

---

2. Initialization Requirements:

REQ201 – The software shall initialize controls in the Unlock state.

3. Diagnosis Requirements:

REQ201 – The software shall determine the lock state of each door based on the lock positions.
- Lock position is under 1mm: Unlock state
- Lock position is over 4mm: Lock state
- Otherwise: Neutral state

REQ202 – The software shall determine the overall vehicle lock state based on all door lock positions.
- All doors in lock position: Lock state
- All doors in unlock position: Unlock state

REQ203 – The software shall determine the overall vehicle lock state to be in failure state due to lock failure in the case where there is no response to a door lock request in under 2 seconds.
Model⇔Code Equality Checks (SIL/PIL, Back 2 Back Test)

**Embedded Coder**

Efficient testing by reuse of model verification test data

Existing data/SLDV generated test data

Model/Code Results Comparison

Model/Code Selection

※ Test automation through Simulink Test.
## Tool Chain Example: Product List

<table>
<thead>
<tr>
<th>Product</th>
<th>Functionality</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Simulink</td>
<td>Modeling: Controller Block</td>
<td>Modeling Module/Integration Test</td>
</tr>
<tr>
<td>Stateflow</td>
<td>Modeling: State Transitions, Flow Charts</td>
<td>Modeling</td>
</tr>
<tr>
<td>Fixed-Point Designer</td>
<td>Modeling: Fixed-Point Processing</td>
<td>Modeling</td>
</tr>
<tr>
<td>Simulink Verification and Validation</td>
<td>Model Coverage Requirements Interface Model Advisor</td>
<td>Module/Integration Test Review and Static Analysis</td>
</tr>
<tr>
<td>Simulink Design Verifier</td>
<td>Property Proving Test Generation Design Error Detection</td>
<td>Review and Static Analysis</td>
</tr>
<tr>
<td>IEC Certification Kit</td>
<td>Traceability Matrix Generation Templates for Certification</td>
<td>ISO26262 Support</td>
</tr>
<tr>
<td>Simulink Report Generator</td>
<td>Report Editing and Generation</td>
<td>Report Generation Model Comparison/Merge</td>
</tr>
</tbody>
</table>
Proving Source Code Correctness
Polyspace Code Prover: Static Code Verification

- **Quality**
  - Prove absence of runtime errors (RTEs)
  - Measure, Improve, Manage

- **Usage**
  - No need to compile, execute, or generate test cases
  - Supports: C/C++/Ada

- **Process**
  - Early detection of RTEs
  - Analyze both hand-code and auto-generated code
  - Measure code reliability

**Quality**
- Prove absence of runtime errors (RTEs)
- Measure, Improve, Manage

**Usage**
- No need to compile, execute, or generate test cases
- Supports: C/C++/Ada

**Process**
- Early detection of RTEs
- Analyze both hand-code and auto-generated code
- Measure code reliability

---

```
static void pointer_arithmetic (void) {
    int array[100];
    int *p = array;
    int i;

    for (i = 0; i < 100; i++) {
        *p = 0;
        p ++;
    }

    if (get_bus_status () > 0) {
        if (get_oil_pressure () > 0) {
            *p = 5;
        } else {
            i ++;
        }
    }
}
```

- **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**

```
variable 'I' (int32): [0 .. 99]
assignment of 'I' (int32): [1 .. 100]
```

---

Analyze all executable paths to detect errors and prove the absence of errors
ISO26262 Functional Safety Standard

- Functional safety standard for automotive equipment
- Based on IEC61508
- Description of purpose and requirements for development
  - Activities for development process (Software safety life cycle)
  - Development and verification tools (Tool qualification)
- Description of new software engineering concepts
  - Model-based development
  - Early verification and validity checks
  - Automatic code generation
Model-Based Design Benefits (ISO26262 excerpt)

Annex B  
(informative)  
ISO/DIS 26262-6

Model-based development

B.1 Objectives

This Annex describes the concept of model-based development of in-vehicle software and outlines its implications on the product development at the software level.

The seamless utilization of models facilitates a highly consistent and efficient development.

ISO/DIS 26262-1

1.74 model-based development

development that uses models to describe the functional behavior of the elements which are to be developed

NOTE or both.  Depending on the level of abstraction used for such a model it can be used for simulation or code generation
MathWorks Solution: Summary
Using Models to Detect Errors Early and Increase Efficiency

- Able to form small V-loops
- Able to detect errors early in the development cycle

- Mode ⇔ Code consistency allows for Simulink simulation results to be considered “truth”.
- Early model verification is possible due to the ability to investigate floating-point models
- Large team development made easy through highly customizable tool chain
- Errors in object code detected easily through synchronization between simulations and SILS/PILS

- Requirements Linking (SLVV)
- Formal Verification (SLDV)
- Model Guideline Check (SLVV)
- Model Coverage Test (SLDV)
- Test Case Auto-Generation (SLDV)
- Report Auto-Generation (Report Generator)
- SILS/PILS (EC)
- Runtime Error Identification (Polyspace)

• Able to form small V-loops
• Able to detect errors early in the development cycle

- Mode ↔ Code consistency allows for Simulink simulation results to be considered “truth”.
- Early model verification is possible due to the ability to investigate floating-point models
- Large team development made easy through highly customizable tool chain
- Errors in object code detected easily through synchronization between simulations and SILS/PILS