Testen zur Absicherung automatisierter Transformationsschritte im Model-Based Design

October 4th 2011

Dr. Marc Segelken
Senior Application Engineer
Agenda

- Testing
  - Existing coverage metrics sufficient for testing?
  - Coverage definition MC/DC
  - Possible blind spots of coverage metrics
  - Coverage definition for enabling observability

- Other means of ensuring correct code generation
  - (Pre-)Qualification of code generator
  - Automatic code reviews

- Summary
Development Process
Verification and Validation in the Design “V”

Requirements

System Design
- Environment
- Physical Components
- Algorithms

System-Level Specification

Component Design

Subsystem Design

Test vectors and expected responses

System Tests

Test vectors and expected outputs

Component Tests

Subsystem Integration & Test

Code Verification and Validation

System-Level Integration & Test

Integration testing

User Acceptance Testing

Continuous V&V

Population ID Boxes:
- Req to Imple
- Imple to Test
- Req to Dsgn
- Req to Test
- Dsgn to Test
- Dsgn to Imple

Requirements Component Design

Test vectors end expected responses

System tests in simulation

Component tests in simulation

Generate

Implementation

Subsystem Implementation

DSP
FPGA
ASIC
Embedded Software
Digital Electronics
C, C++
VHDL, Verilog
MCU, DSP, FPGA, ASIC

System-Level Integration & Test

Integration & Test Complete

User Acceptance Testing
Development Process
Verification and Validation in the Design “V”

Requirements

System Design
- Environment
- Physical Components
- Algorithms

User Acceptance Testing
Complete Integration & Test
System-Level Integration & Test

Component Design

Subsystem Design

System-Level Specification
Subsystem Integration & Test

Component Tests
Test vectors and expected outputs
Run component tests on target
Run system tests on integrated controller
System tests in simulation

System Tests
Test vectors and expected responses

Code Verification and Validation

Implementation
Subsystem Implementation

Integration testing

Integration & Test

System validation/Acceptance Test

System tests in simulation

Development Process
Verification and Validation in the Design “V”

Requirements

System Design
- Environment
- Physical Components
- Algorithms

User Acceptance Testing
Complete Integration & Test
System-Level Integration & Test

Component Design

Subsystem Design

System-Level Specification
Subsystem Integration & Test

Component Tests
Test vectors and expected outputs
Run component tests on target
Run system tests on integrated controller
System tests in simulation

System Tests
Test vectors and expected responses

Code Verification and Validation

Implementation
Subsystem Implementation

Integration testing

System validation/Acceptance Test

System tests in simulation
Existing Coverage Metrics sufficient for Testing?

- Testing
  - Functional: Does the model/code fulfill the requirements?
  - Structural: Is the code functionally equivalent with model?

- Coverage Metrics to define test amount criterion
  - Many metrics available, stronger ones take more testing effort
  - MC/DC is strongest industry standard in Aerospace

- But – is this sufficient?
  - What kind of defects might fall through?
General Approach – Measuring Defect Detection Rate

- Automate introduction of single defects across whole model
  - Ideas: wrong logical/arithmetic function, wrong thresholds, wrong lookup table values, change transition execution order, insert delays, ...

- Then check if each defect is detected automatically by applied test set

- Problems
  - No standardized approach
  - Which defects to be inserted?
  - Huge testing effort for big models ➔ parallel execution needed
Coverage Metrics

- **MC/DC (Modified Condition Decision Coverage)**
  - Each of the conditions a,b,c must be demonstrated to change output of expression
  - ➔ keep value of all conditions but one and observe change in output by toggling one
Problem 1

- Test might fail to use faulty element
  - MC/DC only considers logic/control flow
  - No arithmetics covered
  - Defects in these parts possibly missed

- Approach
  - Using Min Max and lookuptable coverage
  - Extending coverage metrics by reasonable data values
  - Examples:
Problem 2: Observability - Propagation Delay of Errors

How many steps to have impact on output?
Using Tests with demonstrated Event Impact on Outputs

- How to demonstrate impact of an event (possible defect) with unknown propagation delay time?
  - As opposed to MC/DC, not possible to just compare subsequent values after conditions toggle
  - Need to show impact difference between occurrence and non-occurrence of event

- one reference test case per event needed
  - with non-occurrence of event

- Example: Demonstrating Impact of Inputs on Outputs of state-based components
Measuring Input Output Impact Coverage
Creating Deviations
Measuring Coverage - Details
Generating Test Cases Automatically

- Harness for extended coverage measurement can be generated by script
- Test generation with user-defined test objectives then same as MC/DC
- Complexity restrictions of test case generation to be considered
Remarks on Coverage Extensions

- As for every testing process and coverage compromise, defect detection not guaranteed

- Yet defect detection gets more probable with additional coverage criteria in place
Testing Summary

- MC/DC does not guarantee reliable defect detection
- Coverage metrics can be extended by data value range coverage
- Including observability of events
Qualifiable Code Generators – Alternative to Testing?

- Idea: Moving effort almost completely to development phase of code generator

- Increased SW quality supposed to rule out defects in code generator that might lead to errors

- Does this approach guarantee absence of defects/errors?

- Same as for DO-178B: Final responsibility stays with tool user
Other Alternative: Automatic Code Review

- Idea: Automatically check if model and code match structurally and functionally

- ➔ Approach of Simulink Code Inspector
Simulink Code Inspector

Independently verify that source code is traceable to and complies with low-level requirements

- Demonstrate that model and source code match structurally and functionally
- Provide model-to-code and code-to-model traceability information
- Eliminate / reduce manual code reviews for DO-178B software
- Same certification credits as qualified code generator
Simulink Code Inspector Overview

Model and code development

Independent code inspection

Model IR

Code IR

IR transformations

Normalized Model IR

Normalized Code IR

Matching

Code inspection report

Simulink Model

C source code

Embedded Coder
### Simulink Code Inspector Report

**Model Checksum:** 2625869732 1436139376 75727854 1791441049  
**Simulink Version:** 7.8  
**Code Inspection Run On:** 30-Sep-2011 16:25:14  
**Inspected Code Files:** C:\Users\image\AppData\Local\Temp\atemp\sldemo_roll_init_sldemo_roll activités

#### Code Inspection Results: Failed to verify

**Function Interface Verification Results:** Verified

<table>
<thead>
<tr>
<th>Function</th>
<th>Status Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>sldemo_roll_init</td>
<td>Verified</td>
</tr>
<tr>
<td>sldemo_roll_step</td>
<td>Verified</td>
</tr>
</tbody>
</table>

**Model To Code Verification Results:** Failed to verify

<table>
<thead>
<tr>
<th>Status</th>
<th>Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>Failed to verify</td>
<td>Modal objects with status Verified: 37</td>
</tr>
<tr>
<td></td>
<td>Modal objects with status Not processed: 0</td>
</tr>
<tr>
<td></td>
<td>Modal objects with status Partially processed: 0</td>
</tr>
<tr>
<td></td>
<td>Modal objects with status Warning: 1</td>
</tr>
<tr>
<td></td>
<td>Modal objects with status Failed: 0</td>
</tr>
</tbody>
</table>

**Code To Model Verification Results:** Failed to verify

<table>
<thead>
<tr>
<th>Function</th>
<th>Status</th>
<th>Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>sldemo_roll_init</td>
<td>Verified</td>
<td>Lines of code with status Verified: 2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Lines of code with status Not processed: 0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Lines of code with status Partially processed: 0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Lines of code with status Warning: 0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Lines of code with status Failed: 0</td>
</tr>
<tr>
<td>sldemo_roll_step</td>
<td>Failed to verify</td>
<td>Lines of code with status Verified: 16</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Lines of code with status Not processed: 0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Lines of code with status Partially processed: 0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Lines of code with status Warning: 1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Lines of code with status Failed: 0</td>
</tr>
</tbody>
</table>

**Temporary Variable Usage Results:** Verified

<table>
<thead>
<tr>
<th>Function</th>
<th>Status</th>
<th>Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>sldemo_roll_init</td>
<td>Verified</td>
<td>Function does not have any temporary variable declarations</td>
</tr>
</tbody>
</table>

Example for bad case: Deviating model and code
Summary

- Coverage Metrics MC/DC can be complemented with non-standard user-defined coverages for data values and observability enhancements

- Automatic Test Case Generation can be used if applicable

- Automatic code reviews can give qualification credit for code generator, eliminating/reducing code reviews