Model-Based Design for Safety-Critical and Mission-Critical Applications

Bill Potter
Technical Marketing
May 2, 2008
Safety-Critical Model-Based Design Workflow

- **Validate**
  - Simulink® & Stateflow®
  - Real-Time Workshop® & Embedded Coder™

- **Trace:**
  - Model/Code Trace Report
  - RMI

- **Conformance:**
  - Model Advisor
  - PolySpace™ Products

- **Verify:**
  - SystemTest™
  - SLDV Property Proving Model Coverage
  - SLDV Test Generation Embedded IDE Link XXX

- **Source Code**
  - Embedded IDE

- **Object Code**

MathWorks
Aerospace and Defence Conference ’08
Requirements Process for Model-Based Design

- Functional, operational, and safety requirements
  - Exist one level above the model
  - Models trace to requirements
- Requirements validation - complete and correct
  - Simulation is a validation technique
  - Traceability can identify incomplete requirements
  - Model coverage can identify incomplete requirements
- Requirements based test cases
  - Test cases trace to requirements
Simulation example – controller and plant
<table>
<thead>
<tr>
<th>ID</th>
<th>Requirement Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>43</td>
<td>[Simulink reference: attitude_controller/Rate Limit (Saturate)]</td>
</tr>
<tr>
<td>20</td>
<td><strong>2.4 Integral Control and Limit</strong></td>
</tr>
<tr>
<td></td>
<td>The integral control shall generate a surface command based on the attitude rate error</td>
</tr>
<tr>
<td></td>
<td>computed by the rate control, integral error gain and the autopilot engage state.</td>
</tr>
<tr>
<td></td>
<td>The total integral command shall be limited to not exceed the integral command limit.</td>
</tr>
<tr>
<td></td>
<td>When the autopilot is not engaged, the integral command and internal state shall be</td>
</tr>
<tr>
<td></td>
<td>held at zero.</td>
</tr>
<tr>
<td>63</td>
<td>[Simulink reference: attitude_controller_harness/Signal Builder (SubSystem)]</td>
</tr>
<tr>
<td>39</td>
<td>[Simulink reference: attitude_controller/Int Gain (Gain)]</td>
</tr>
<tr>
<td>38</td>
<td>[Simulink reference: attitude_controller/Not engaged (Logic)]</td>
</tr>
<tr>
<td>37</td>
<td>[Simulink reference: attitude_controller/Integrator (DiscreteIntegrator)]</td>
</tr>
<tr>
<td>21</td>
<td><strong>2.5 Surface Command and Limit</strong></td>
</tr>
</tbody>
</table>
Requirements trace example – view from Simulink to DOORS

Chapter 2. System - attitude_controller

![Diagram of attitude controller system]

Table 2.1. Block Requirements Table

<table>
<thead>
<tr>
<th>Name</th>
<th>Requirements</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cmd Limit</td>
<td>2.1.2 Parameters, 2.5 Surface Command and Limit</td>
</tr>
<tr>
<td>Disp Gain</td>
<td>2.1.2 Parameters, 2.2 Attitude Control and Limit</td>
</tr>
<tr>
<td>Disp Limit</td>
<td>2.1.2 Parameters, 2.2 Attitude Control and Limit</td>
</tr>
<tr>
<td>Disp_Cord</td>
<td>2.1.1 Inputs</td>
</tr>
</tbody>
</table>
Requirements based test trace example – view from Simulink Signal Builder block to DOORS
**Model coverage report example**

### Discrete integrator block "Integrator"

**Parent:** `altitude_controller`

<table>
<thead>
<tr>
<th>Metric</th>
<th>Coverage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cyclomatic Complexity</td>
<td>3</td>
</tr>
<tr>
<td>Decision (C1)</td>
<td>100% (SASE) decision outcomes</td>
</tr>
</tbody>
</table>

**Decisions analyzed:**

<table>
<thead>
<tr>
<th></th>
<th>100%</th>
<th>100%</th>
<th>100%</th>
<th>100%</th>
<th>100%</th>
<th>100%</th>
<th>100%</th>
<th>100%</th>
<th>100%</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reset</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>false</td>
<td>201401</td>
<td>201401</td>
<td>201401</td>
<td>201401</td>
<td>201401</td>
<td>201401</td>
<td>201401</td>
<td>201401</td>
<td>201401</td>
</tr>
<tr>
<td>true</td>
<td>200401</td>
<td>200401</td>
<td>200401</td>
<td>200401</td>
<td>200401</td>
<td>200401</td>
<td>200401</td>
<td>200401</td>
<td>200401</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>100%</th>
<th>100%</th>
<th>100%</th>
<th>100%</th>
<th>100%</th>
<th>100%</th>
<th>100%</th>
<th>100%</th>
<th>100%</th>
</tr>
</thead>
<tbody>
<tr>
<td>integration result &lt;= lower limit</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>false</td>
<td>401401</td>
<td>401401</td>
<td>401401</td>
<td>401401</td>
<td>401401</td>
<td>401401</td>
<td>401401</td>
<td>401401</td>
<td>401401</td>
</tr>
<tr>
<td>true</td>
<td>0401</td>
<td>0401</td>
<td>0401</td>
<td>0401</td>
<td>0401</td>
<td>0401</td>
<td>0401</td>
<td>0401</td>
<td>0401</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>100%</th>
<th>100%</th>
<th>100%</th>
<th>100%</th>
<th>100%</th>
<th>100%</th>
<th>100%</th>
<th>100%</th>
<th>100%</th>
</tr>
</thead>
<tbody>
<tr>
<td>integration result &gt;= upper limit</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>false</td>
<td>401401</td>
<td>401401</td>
<td>401401</td>
<td>401401</td>
<td>401401</td>
<td>401401</td>
<td>401401</td>
<td>401401</td>
<td>401401</td>
</tr>
<tr>
<td>true</td>
<td>0401</td>
<td>0401</td>
<td>0401</td>
<td>0401</td>
<td>0401</td>
<td>0401</td>
<td>0401</td>
<td>0401</td>
<td>0401</td>
</tr>
</tbody>
</table>

**Logic block "Not engaged"**

**Parent:** `altitude_controller`

<table>
<thead>
<tr>
<th>Metric</th>
<th>Coverage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cyclomatic Complexity</td>
<td>0</td>
</tr>
<tr>
<td>Condition (C1)</td>
<td>100% (2/2) condition outcomes</td>
</tr>
</tbody>
</table>
Requirements Process take-aways

- Early requirements validation
  - Eliminates rework typically seen at integration on projects with poor requirements
- Early test case development
  - Validated requirements are complete and verifiable which results in well defined test cases
- Requirements management and traceability
  - Requirements management interfaces provide traceability for design and test cases
Design Process for Model-Based Design

- Model-Based Design
  - Create the design - Simulink and Stateflow®
  - Modular design for teams - Model Reference
  - Model architecture/regression analysis - Model Dependency Viewer
  - Documented design - Simulink Report Generator
  - Requirements traceability using Simulink Verification and Validation™
  - Design conforms to standards using Model Advisor
Example detailed design including model reference and subsystems

Top Model

Subsystem

Reference Model
Model dependency viewer
Design Verification for Model-Based Design

- Requirements based test cases
  - Automated testing using SystemTest™ and Simulink Verification and Validation
  - Traceability using Simulink Verification and Validation
- Robustness testing and analysis
  - Built in Simulink run-time diagnostics
  - Formal proofs using Simulink Design Verifier™
- Coverage Analysis
  - Verify structural coverage of model
  - Verify data coverage of model
SystemTest for requirements based testing
SystemTest – example report

Data Plotting and expected results comparisons

Summary of results
Signal Builder and Assertion Blocks
Model coverage report example – signal ranges

![Coverage Report](image)

<table>
<thead>
<tr>
<th>Hierarchy</th>
<th>Min</th>
<th>Max</th>
<th>Min</th>
<th>Max</th>
<th>Min</th>
<th>Max</th>
<th>Min</th>
<th>Max</th>
<th>Min</th>
<th>Max</th>
<th>Min</th>
<th>Max</th>
</tr>
</thead>
<tbody>
<tr>
<td>attitude_controller</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>. . . . . . . . . . .</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Integrator</td>
<td>-1</td>
<td>9</td>
<td>-10</td>
<td>9</td>
<td>-1</td>
<td>1</td>
<td>-4</td>
<td>4</td>
<td>-5</td>
<td>5</td>
<td>-1</td>
<td>1</td>
</tr>
<tr>
<td>Not engaged</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Cmd Limit</td>
<td>-1</td>
<td>9</td>
<td>-10</td>
<td>9</td>
<td>-1</td>
<td>1</td>
<td>-4</td>
<td>4</td>
<td>-5</td>
<td>5</td>
<td>-1</td>
<td>1</td>
</tr>
<tr>
<td>Disp Limit</td>
<td>-1</td>
<td>9</td>
<td>-10</td>
<td>9</td>
<td>-1</td>
<td>1</td>
<td>-4</td>
<td>4</td>
<td>-5</td>
<td>5</td>
<td>-1</td>
<td>1</td>
</tr>
<tr>
<td>Rate Limit</td>
<td>-1</td>
<td>9</td>
<td>-10</td>
<td>9</td>
<td>-1</td>
<td>1</td>
<td>-4</td>
<td>4</td>
<td>-5</td>
<td>5</td>
<td>-1</td>
<td>1</td>
</tr>
<tr>
<td>Disp Gain</td>
<td>-1</td>
<td>9</td>
<td>-10</td>
<td>9</td>
<td>-1</td>
<td>1</td>
<td>-4</td>
<td>4</td>
<td>-6</td>
<td>6</td>
<td>-1</td>
<td>1</td>
</tr>
<tr>
<td>Int Gain</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Rate Gain</td>
<td>-1</td>
<td>9</td>
<td>-10</td>
<td>9</td>
<td>-1</td>
<td>1</td>
<td>-4</td>
<td>4</td>
<td>-5</td>
<td>5</td>
<td>-1</td>
<td>1</td>
</tr>
<tr>
<td>Sum</td>
<td>-1</td>
<td>9</td>
<td>-10</td>
<td>9</td>
<td>-1</td>
<td>1</td>
<td>-1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Sum1</td>
<td>-1</td>
<td>9</td>
<td>-10</td>
<td>9</td>
<td>-1</td>
<td>1</td>
<td>-1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Sum2</td>
<td>-1</td>
<td>9</td>
<td>-10</td>
<td>9</td>
<td>-1</td>
<td>1</td>
<td>-1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

MathWorks
Aerospace and Defence Conference ’08
Simulink Design Verifier – Coverage Test

Model

Test Report

Generated Test Cases

Test Case 7

Summary
Length: 0.13 Seconds (6 sample periods)
Objective Count: 10

Objectives Reached At:

<table>
<thead>
<tr>
<th>Step</th>
<th>Time</th>
<th>Objectives</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>0.01</td>
<td>7</td>
</tr>
<tr>
<td>3</td>
<td>0.02</td>
<td>11</td>
</tr>
<tr>
<td></td>
<td></td>
<td>16</td>
</tr>
<tr>
<td></td>
<td></td>
<td>18</td>
</tr>
<tr>
<td>9</td>
<td>0.08</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td></td>
<td>12</td>
</tr>
<tr>
<td></td>
<td></td>
<td>14</td>
</tr>
<tr>
<td>13</td>
<td>0.12</td>
<td>15</td>
</tr>
</tbody>
</table>

Generated Input Data:

<table>
<thead>
<tr>
<th>Time 0</th>
<th>0.01</th>
<th>0.02</th>
<th>0.07</th>
<th>0.08</th>
</tr>
</thead>
<tbody>
<tr>
<td>Step</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>Raw</td>
<td>1.28</td>
<td>1</td>
<td>1.28</td>
<td>0</td>
</tr>
</tbody>
</table>
Simulink Design Verifier – Objective Test

Model with Constraints and Objectives

Test Report

Chapter 3. Test Cases / Counterexamples

Test Case 1

Summary
Length: 0.13 Seconds (4 sample periods)
Objective Count: 2
Objectives Reached At:

<table>
<thead>
<tr>
<th>Step</th>
<th>Time</th>
<th>Objectives</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>0.06</td>
<td>2</td>
</tr>
<tr>
<td>13</td>
<td>0.12</td>
<td>1</td>
</tr>
</tbody>
</table>

Generated Input Data:

<table>
<thead>
<tr>
<th>Step</th>
<th>Time</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0.01</td>
<td>0.07</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>raw</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

Aerospace and Defence Conference ’08
Simulink Design Verifier – Property Proving

Model with Assumption and Objective

Property to be proven

Report

Chapter 2. Test/Proof Objectives

Table of Contents

Status

Verify True Output

Objectives of: Assertion
Design Process take-aways

- Modular reusable implementations
  - Platform independent design
  - Scalable to large teams
- Consistent and compliant implementations
  - Common design language
  - Automated verification of standards compliance
- Efficient verification process
  - Develop verification procedures in parallel with design
  - Coverage analysis early in the process
  - Automated testing and analysis
Coding Process for Model-Based Design

- Automatic code generation
  - Real-Time Workshop Embedded Coder
- Traceability
  - HTML Code Traceability Report
- Source code verification
  - Complies with standards using PolySpace MISRA-C® checker
  - Accurate, consistent and robust using PolySpace™ verifier
Incrementally Generate Code

- Incremental code generation is supported via Model Reference.
- When a model is changed, only models depending on it are subject to regeneration of their code.

- Reduces application build times and ensure stability of a project’s code.
- Degree of dependency checking is configurable.
Add Links to Requirements

Requirements appear in the code

```matlab
94  /* DiscretePulseGenerator: '<Root>/clock'
95    *
96    * Requirements for '<Root>/clock':
97    * 1. Clock period shall be consistent with chirp tolerance
98    */
99
100   rth_clock =
101       (rtDWork.clockTickCounter < 1.0 &&
102         rtDWork.clockTickCounter >= 0) ?
103         1.0 ;
104         0.0;
105       if (rtDWork.clockTickCounter >= 2.0-1) {
```
Code to Model Trace Report
Compliance history of generated code

- Our MISRA-C test suite consists of several example models
- Results shown for most frequently violated rules

- Improving MISRA-C compliance with each release, e.g.
  - Eliminate Stateflow `goto` statements (R2007a)
  - Compliant parentheses option available (R2006b)
  - Generate `default` case for `switch-case` statements (R2006b)

- MathWorks MISRA-C Compliance Package available upon request http://www.mathworks.com/support/solutions/data/1-1FP0W.html
Simulink Integration with PolySpace Products

- **Input1**
  - Entries
  - Varying from -500 to 500

- **K1 and K2**
  - Constants
  - Can be tuned from -297 to 303

- **Math operations**
  - Divide, add, min/max, product, subtract, sum...

- **Lookup tables**
  - Maps, surfaces, algorithms, extrapolations
  - Adjusted, tuned

- **Division by Zero?**

- **Overflow?**
See results in the model

- Change the model
- Generate the production code
- Run PolySpace software

PolySpace detected an error here
(after having analyzed the generated code)
Coding Process takeaways

- Reusable and platform independent source code
- Traceability
- MISRA-C compliance
- Static verification and analysis
Integration Process for Model-Based Design

- Executable object code generation
  - ANSI® or ISO® C or C++ compatible compiler
  - Run-time libraries provided
- Executable object code verification
  - Test generation using Simulink Design Verifier
  - Capability to build interface for Processor-In-the-Loop (PIL) testing
  - Analyze code coverage during PIL
  - Analyze execution time during PIL
  - Analyze stack PIL
Processor-in-the-Loop (PIL) Verification
- Execute Generated Code on Target Hardware

**Execution**
- on host and target
- non-real-time

**Communication via one of**
- data link e.g. serial, CAN, TCP/IP
- debugger integration with MATLAB
Integration Process Takeaways

- Integration with multiple development environments
- Test cases and harnesses generated automatically
- Efficient processor in-the-loop test capability
Wrap-up

- Tools to support the entire safety critical development process
- Participation on SC-205/WG-71 committee for DO-178C
- Safety-Critical/DO-178B guideline document
  - Available to licensed customers with Real-Time Workshop Embedded Coder
  - Contact Bill Potter (bill.potter@mathworks.com) or Tom Erkkinen (tom.erkkinen@mathworks.com)