Model-Based Design for Safety Critical Applications

Bill Potter
The MathWorks
Attributes of Safety Critical Systems

- Reliably perform intended function
- Contain no unintended function
- Implemented with redundancy
- Contain fault detection
- Robust design
- Robust code
Attributes of Safety Critical Process

- Complete and correct requirements
- Design standards are applied
- Coding standards are applied
- Bi-directional traceability
- Requirements based testing
- Robustness verification
- Coverage analysis
- Safety Analysis
- Failure Modes and Effects Analysis (FMEA)
Safety-Critical Model-Based Design Workflow and Activities

**Goals**
- Requirements Document
- Plant model
- Complete
- Correct
- Test cases
- Safety Analysis

**Design Process**
- Controller design
- Correct
- Robust
- Traceable
- Conforms to standards
- FMEA

**Coding Process**
- Generated code
- Correct
- Robust
- Traceable
- Conforms to standards

**Integration Process**
- Compiled code
- Correct
- Robust
- Coverage Analysis

**Requirements Process**
- DOORS, TCE, MSWord, etc
- MATLAB®
- Simulink®
- Stateflow®
- Simulink Report Generator

**Generated Source Code**
- Real-Time Workshop® Embedded Coder
- PolySpace Verifier
- Simulink Report Generator

**Executable Object Code**
- Software-Software integration
- Hardware-Software integration
- Processor in-the-loop
- SystemTest
- Simulink Report Generator

**Hardware**

**Controller Design**
- MATLAB
- Simulink/ Stateflow
- Simulink Verification & Validation
- Simulink Design Verifier
- SystemTest
- Simulink Report Generator

**Simulate to Verify Design**

**Simulate prototype to Validate Requirements**

**Requirements**
Requirements Process for Model-Based Design

- Functional, operational, and safety requirements
  - Exist one level above the model
  - Models trace to requirements

- Requirements validation
  - Prove requirements are complete and correct
  - Simulation is a validation technique
  - Traceability can identify incomplete requirements
  - Model coverage can identify incomplete requirements

- Requirements based test cases
  - Traceability of tests to requirements
Simulation example – controller and plant
## Requirements trace example – view from DOORS to Simulink

<table>
<thead>
<tr>
<th>ID</th>
<th>Software requirements for a reusable attitude controller</th>
</tr>
</thead>
<tbody>
<tr>
<td>43</td>
<td>[Simulink reference: attitude_controller/Rate Limit (Saturate)]</td>
</tr>
<tr>
<td>20</td>
<td>2.4 Integral Control and Limit</td>
</tr>
<tr>
<td></td>
<td>The integral control shall generate a surface command based on the attitude rate error computed by the rate control, integral error gain and the autopilot engage state. The total integral command shall be limited to not exceed the integral command limit. When the autopilot is not engaged, the integral command and internal state shall be 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>2.5 Surface Command and Limit</td>
</tr>
</tbody>
</table>
Requirements trace example – view from Simulink to DOORS

Chapter 2. System - attitude_controller

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</td>
</tr>
<tr>
<td></td>
<td>2.6 Surface Command and Limit 036000022 #23</td>
</tr>
<tr>
<td>Disp Gain</td>
<td>2.1.2 Parameters</td>
</tr>
<tr>
<td></td>
<td>2.2 Attitude Control and Limit 036000022 #16</td>
</tr>
<tr>
<td>Disp Limit</td>
<td>2.1.2 Parameters</td>
</tr>
<tr>
<td></td>
<td>2.2 Attitude Control and Limit 036000022 #16</td>
</tr>
<tr>
<td>Disp_Cmd</td>
<td>2.1.1 Inputs 036000022 #22</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:** attitude_controller  

**Metric**  
- **Coverage**
- **Cyclomatic Complexity:** 3  

**Decision (D1):** 100% (5/5) decision outcomes

**Decisions analyzed:**

<table>
<thead>
<tr>
<th>Reset</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>
<th>100%</th>
<th>100%</th>
<th>100%</th>
<th>100%</th>
</tr>
</thead>
<tbody>
<tr>
<td>false</td>
<td>201/401</td>
<td>201/401</td>
<td>201/401</td>
<td>201/401</td>
<td>201/401</td>
<td>201/401</td>
<td>201/401</td>
<td>201/401</td>
<td>201/401</td>
<td>201/401</td>
<td>201/401</td>
<td>201/401</td>
<td>201/401</td>
</tr>
</tbody>
</table>

**integration result <= lower limit**

| false | 50% | 50% | 50% | 50% | 50% | 50% | 50% | 50% | 50% | 50% | 100% | 100% |
| true  | 0/401 | 0/401 | 0/401 | 0/401 | 0/401 | 0/401 | 0/401 | 0/401 | 0/401 | 0/401 | 59/401 | 59/401 |

**integration result >= upper limit**

| false | 50% | 50% | 50% | 50% | 50% | 50% | 50% | 50% | 50% | 50% | 100% | 100% |
| true  | 0/401 | 0/401 | 0/401 | 0/401 | 0/401 | 0/401 | 0/401 | 0/401 | 0/401 | 0/401 | 59/401 | 59/401 |

#### Logic block "Not engaged"

**Parent:** attitude_controller  

**Metric**  
- **Coverage**
- **Cyclomatic Complexity:** 0  

**Condition (C1):** 100% (2/2) condition outcomes
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

- Conformance to standards
  - Design conforms to standards – Model Advisor

- Traceability
  - Design to requirements - Requirements Management Interface
Example detailed design including model reference and subsystems
Model dependency viewer

Dependency view: do178b_dhc2
02-May-2007 10:26:59

- Library
- Model

Graph:
- do178b_dhc2
  - pitch_ap
  - roll_ap
  - yaw_damper
    - Altitude_Mode
    - attitude_controller
    - Heading_Mode
Example Model Advisor report

Model Advisor Report for 'attitude_controller'

Model version: 1.21
Generated on: 08-May-2007 10:35:27
42 of 45 Passed

- Check model, local libraries, and referenced models for known upgrade issues
  Passed

- Identify unconnected lines, input ports, and output ports
  Passed

- Check root model import block specifications
  Passed

- Check solver for code generation
  Sample times for this model is Unconstrained. If the model does not specify any sample times, consider setting its Periodic sample time constraint parameter to Ensure sample time independent. Otherwise, set the parameter to Specified if the model will not be referenced.

- Identify questionable blocks within the specified system
  Check for blocks not supported by Real-Time Workshop
Design Verification for Model-Based Design

- Requirements based test cases
  - Automated testing using SystemTest/Simulink V&V
  - Traceability using Requirements Management Interface
  - Capability to inject faults for FMEA
- 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

### Signal Ranges:

<table>
<thead>
<tr>
<th>Hierarchy</th>
<th>Test 1</th>
<th>Test 2</th>
<th>Test 3</th>
<th>Test 4</th>
<th>Test 5</th>
<th>Test 6</th>
<th>Test 7</th>
<th>Test 8</th>
<th>Test 9</th>
<th>Test 10</th>
<th>Overall</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Min</td>
<td>Max</td>
<td>Min</td>
<td>Max</td>
<td>Min</td>
<td>Max</td>
<td>Min</td>
<td>Max</td>
<td>Min</td>
<td>Max</td>
<td>Min</td>
</tr>
<tr>
<td>attitude_controller</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>. . . Integrator</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>-3</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>1</td>
</tr>
<tr>
<td>. . . Cmd Limit</td>
<td>-1</td>
<td>1</td>
<td>-9</td>
<td>9</td>
<td>-10</td>
<td>10</td>
<td>-1</td>
<td>1</td>
<td>-4</td>
<td>4</td>
<td>-5</td>
</tr>
<tr>
<td>. . . Disp Limit</td>
<td>-1</td>
<td>1</td>
<td>-9</td>
<td>9</td>
<td>-10</td>
<td>10</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>. . . Rate Limit</td>
<td>-1</td>
<td>1</td>
<td>-9</td>
<td>9</td>
<td>-10</td>
<td>10</td>
<td>-1</td>
<td>1</td>
<td>-4</td>
<td>4</td>
<td>-5</td>
</tr>
<tr>
<td>. . . Disp Gain</td>
<td>-1</td>
<td>1</td>
<td>-9</td>
<td>9</td>
<td>-10</td>
<td>10</td>
<td>-1</td>
<td>1</td>
<td>-4</td>
<td>4</td>
<td>-6</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>
</tr>
<tr>
<td>. . . Rate Gain</td>
<td>-1</td>
<td>1</td>
<td>-9</td>
<td>9</td>
<td>-10</td>
<td>10</td>
<td>-1</td>
<td>1</td>
<td>-4</td>
<td>4</td>
<td>-5</td>
</tr>
<tr>
<td>. . . Sum</td>
<td>-1</td>
<td>1</td>
<td>-9</td>
<td>9</td>
<td>-10</td>
<td>10</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>1</td>
<td>-9</td>
<td>9</td>
<td>-10</td>
<td>10</td>
<td>-1</td>
<td>1</td>
<td>-4</td>
<td>4</td>
<td>-5</td>
</tr>
<tr>
<td>. . . Sum2</td>
<td>-1</td>
<td>1</td>
<td>-9</td>
<td>9</td>
<td>-10</td>
<td>10</td>
<td>-1</td>
<td>1</td>
<td>-4</td>
<td>4</td>
<td>-5</td>
</tr>
</tbody>
</table>

MathWorks
Aerospace and Defense Conference ’07
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>0</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>5, 11, 16, 18</td>
</tr>
<tr>
<td>9</td>
<td>0.08</td>
<td>10, 12, 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</th>
<th>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>
<td>5</td>
</tr>
<tr>
<td>raw</td>
<td>128</td>
<td>1</td>
<td>128</td>
<td>0</td>
<td>128</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>Time</th>
<th>0.01</th>
<th>0.07</th>
</tr>
</thead>
<tbody>
<tr>
<td>Step</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>raw</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>
Design Process take-aways

- Modular reusable implementations
  - Platform independent design and code
  - 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
  - Automated analysis techniques
  - Coverage analysis early in the process
Coding Process for Model-Based Design

- Incremental code generation
  - Model Reference
- Traceability
  - HTML Code 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

```
/* DiscretePulseGenerator: '<Root>/clock' */
/* Requirements for '<Root>/clock': */
/* 1. Clock period shall be consistent with chirp tolerance */
rtb_clock =
(rtDWork.clockTickCounter < 1.0 &
  rtDWork.clockTickCounter >= 0) ?
  1.0 :
  0.0;
if (rtDWork.clockTickCounter >= 2.0-1) {
```

Requirements appear in the code
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/1-1IFP0W.html

MathWorks Aerospace and Defense Conference ’07
Coding Process take-aways

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

- Executable object code generation
  - ANSI/ISO C or C++ compatible compiler
  - Makefile generation capability
  - Run-time libraries provided

- Executable object code verification
  - Capability to build interface for Processor-In-the-Loop (PIL) testing
  - Analyze code coverage during PIL
  - Analyze execution time during 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 take-aways

- Integration with multiple development environments
- Efficient processor in-the-loop test capability
Wrap-up

- Tools to support the entire safety critical development process
  - Requirements
  - Design
  - Code
  - Executable
  - Verification
- MathWorks is participating on SC-205/WG-71 committee which is working on Revision C of DO-178
- See the various demos in the exhibit area