Model-Based Design for Safety Critical Applications

Bill Potter
The MathWorks

MathWorks
Aerospace and Defense Conference '07

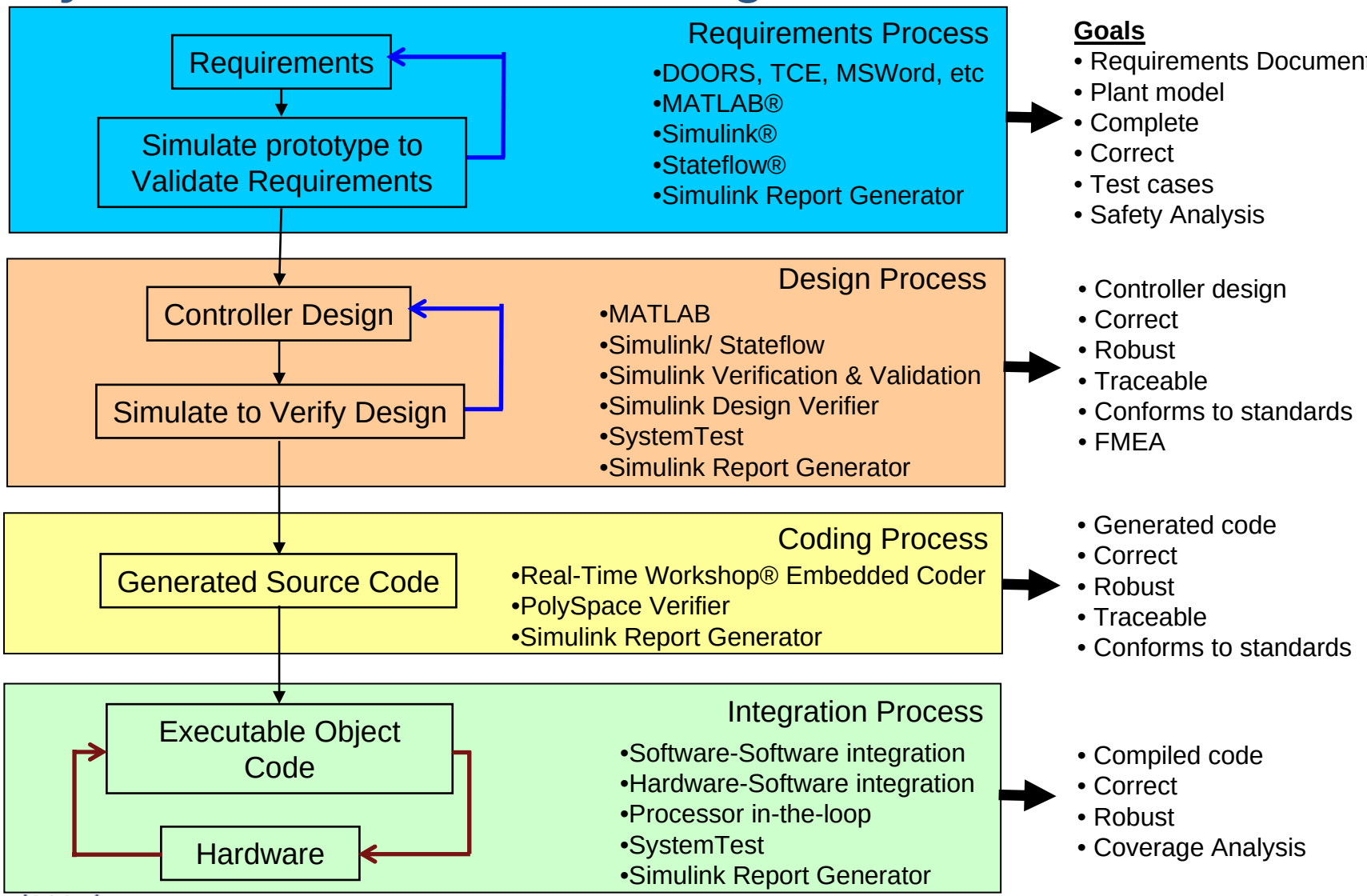
Attributes of Safety Critical Systems

- 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)

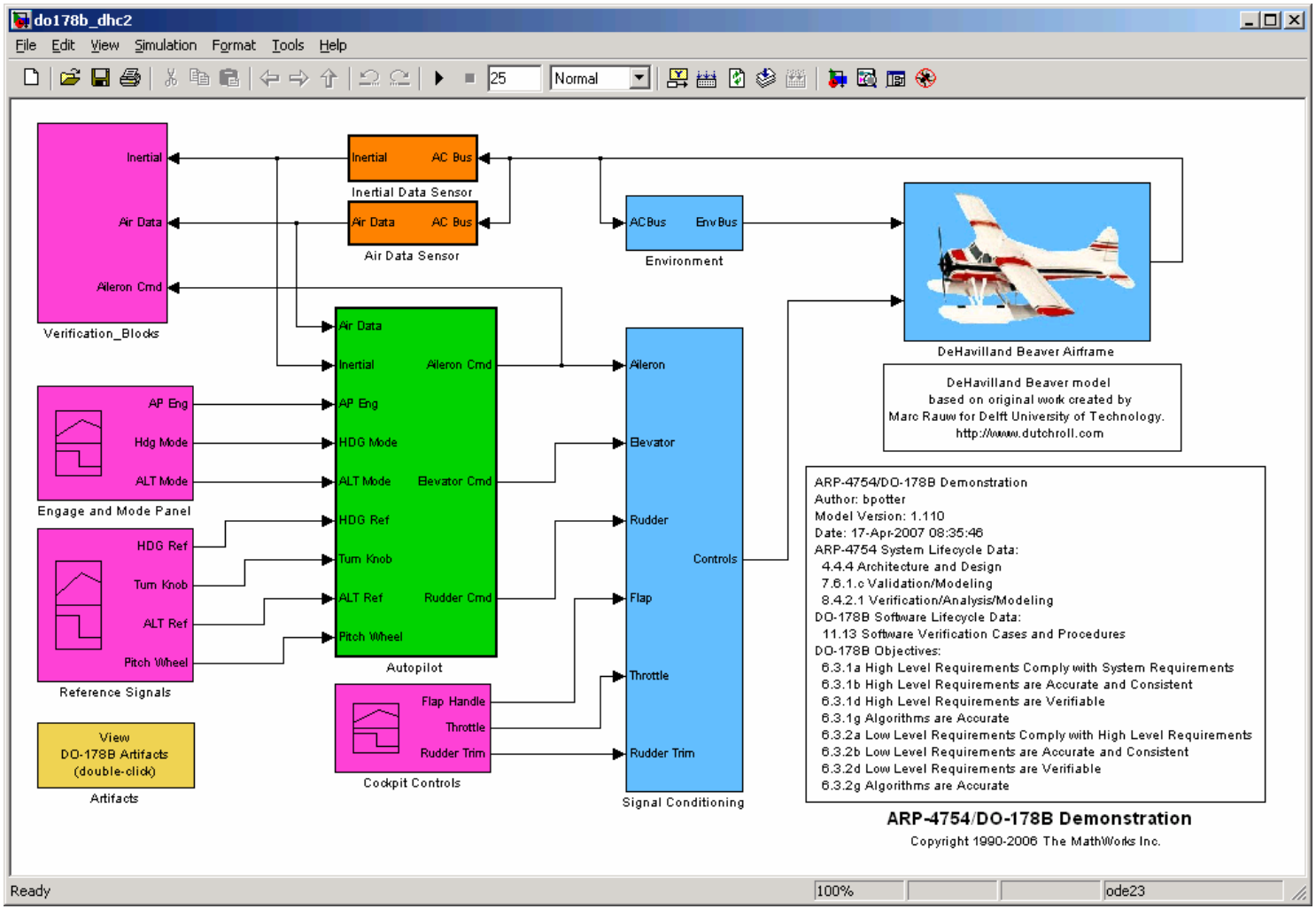
Safety-Critical Model-Based Design Workflow and Activities



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

Formal module '/Autopilot Project/Attitude Controller Derived Requirements' current 0.0 - DOORS

File Edit View Insert Link Analysis Table Tools User MATLAB Help

Standard view | All levels

- Attitude Controller Derive
  - 1 Introduction
  - 2 Component Desig

ID	Software requirements for a reusable attitude controller
43	[Simulink reference: attitude_controller/Rate Limit (Saturate)]
20	**2.4 Integral Control and Limit** The integral control shall generate a surface command based on the attitude rate error computed by the rate control, integral error gain and the autompilot 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.
63	[Simulink reference: attitude_controller_harness/Signal Builder (SubSystem)]
39	[Simulink reference: attitude_controller/Int Gain (Gain)]
38	[Simulink reference: attitude_controller/Not engaged (Logic)]
37	[Simulink reference: attitude_controller/Integrator (DiscreteIntegrator)]
21	2.5 Surface Command and Limit

Username: bpotter | Exclusive edit mode

Requirements trace example – view from Simulink to DOORS

attitude_controller Model Requirements Report

File Edit View Go Debug Desktop Window Help

Location: file:///C:/local_work_area/demos/autopilot_R2007a_mdlref/attitude_controller_reqreport.html

Chapter 2. System - attitude_controller

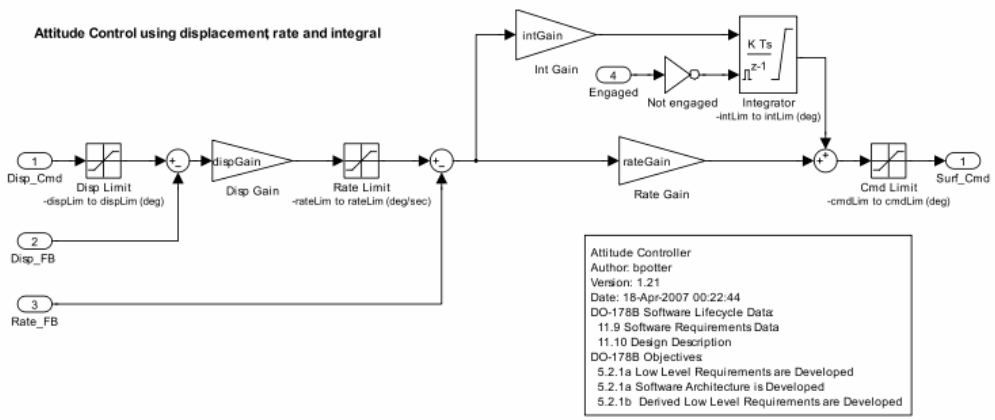
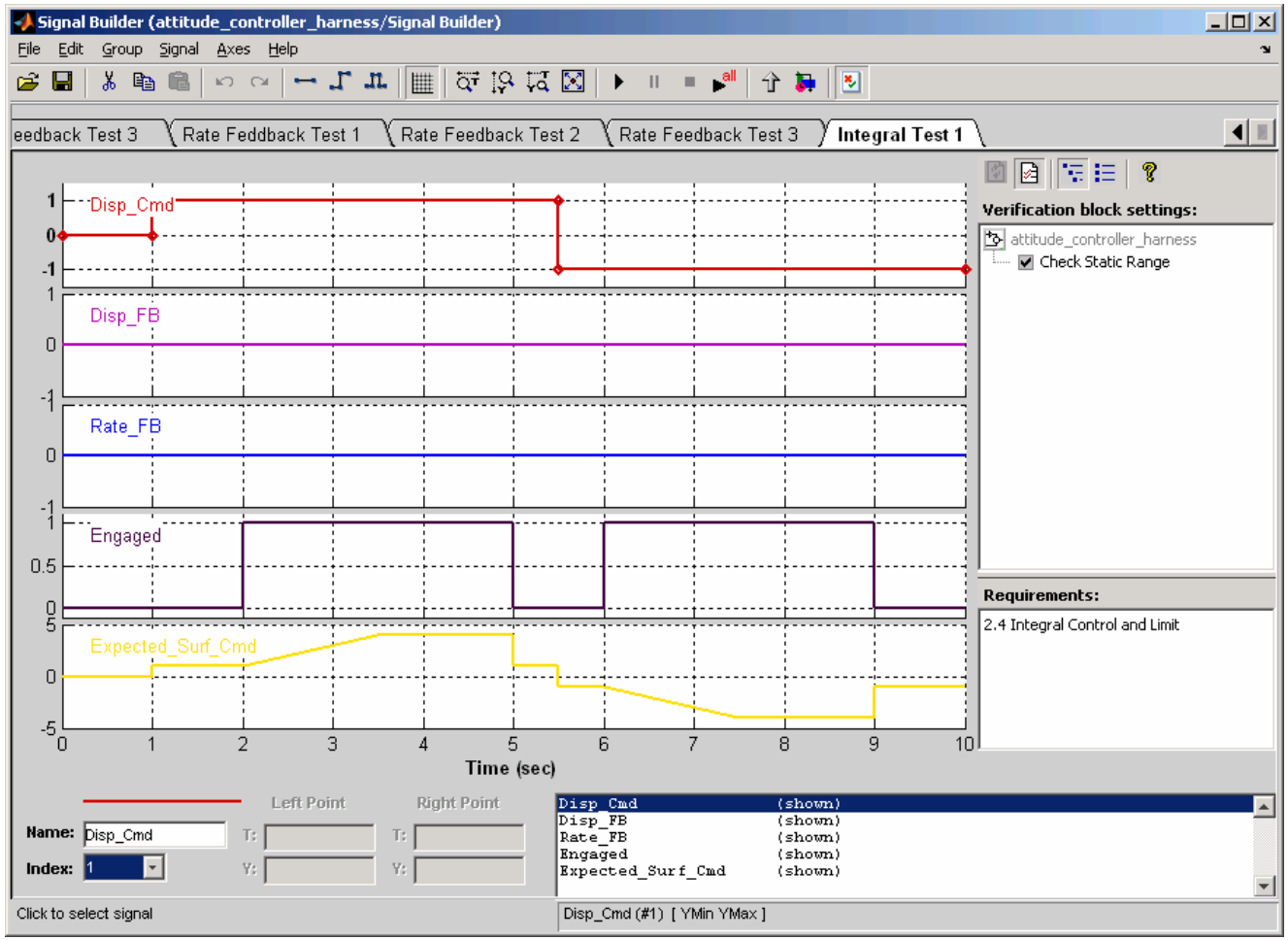


Table 2.1. Block Requirements Table

Name	Requirements
Cmd Limit	2.1.2 Parameters 00000022 #23 2.5 Surface Command and Limit 00000022 #21
Disp Gain	2.1.2 Parameters 00000022 #23 2.2 Attitude Control and Limit 00000022 #16
Disp Limit	2.1.2 Parameters 00000022 #23 2.2 Attitude Control and Limit 00000022 #16
Disp_Cmd	2.1.1 Inputs 00000022 #22

Requirements-based test trace example – view from Simulink Signal Builder block to DOORS



Model coverage report example

attitude_controller Coverage Report

File Edit View Go Debug Desktop Window Help

Location: file:///C:/local_work_area/demos/autopilot_R2007a_mdlref/attitude_control_results19486407.html

Discrete integrator block "Integrator"

Parent: /attitude_controller

Metric	**Coverage**
Cyclomatic Complexity	3
Decision (D1)	100% (6/6) decision outcomes

Decisions analyzed:

Reset	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%
false	201/401	201/401	201/401	201/401	201/401	201/401	201/401	201/401	201/401	240/401	2049/4010
true	200/401	200/401	200/401	200/401	200/401	200/401	200/401	200/401	200/401	161/401	1961/4010
integration result <= lower limit	50%	50%	50%	50%	50%	50%	50%	50%	50%	100%	100%
false	401/401	401/401	401/401	401/401	401/401	401/401	401/401	401/401	401/401	283/342	3892/3951
true	0/401	0/401	0/401	0/401	0/401	0/401	0/401	0/401	0/401	59/342	59/3951
integration result >= upper limit	50%	50%	50%	50%	50%	50%	50%	50%	50%	100%	100%
false	401/401	401/401	401/401	401/401	401/401	401/401	401/401	401/401	401/401	342/401	3951/4010
true	0/401	0/401	0/401	0/401	0/401	0/401	0/401	0/401	0/401	59/401	59/4010

Logic block "Not engaged"

Parent: /attitude_controller

Metric	**Coverage**
Cyclomatic Complexity	0
Condition (C1)	100% (2/2) condition outcomes

Done

Requirements Process take-aways

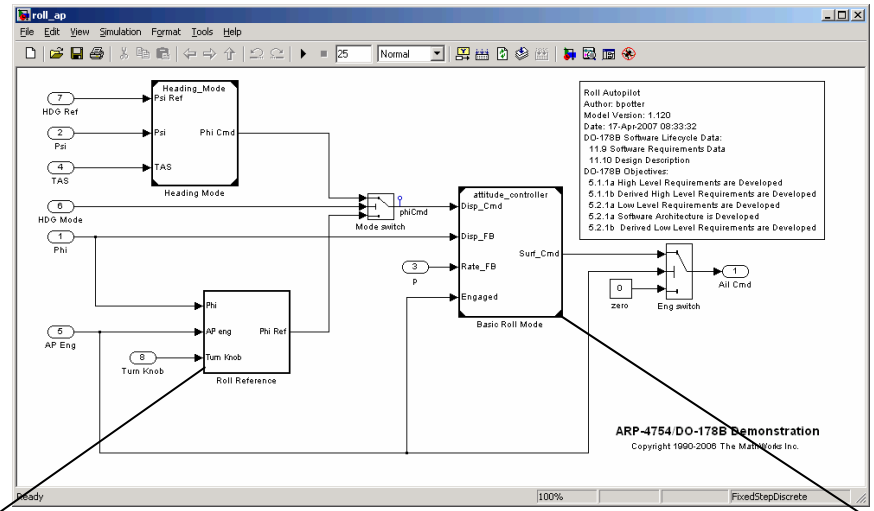
- Early requirements validation
  - Eliminates rework typically seen at integration on projects with poor requirements
- Early test case development
  - Validated requirements are complete and verifiable which results in well defined test cases
- Requirements management and traceability
  - Requirements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

Design Process for Model-Based Design

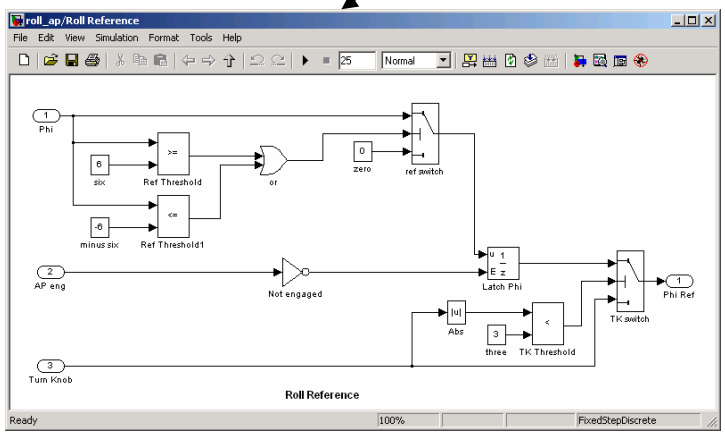
- Model-Based Design
  - Create the design - Simulink and Stateflow
  - Modular design for teams - Model Reference
  - Model architecture/regression analysis - Model Dependency Viewer
  - Documented design - Simulink Report Generator
- Conformance to standards
  - Design conforms to standards – Model Advisor
- Traceability
  - Design to requirements - Requirements Management Interface

Example detailed design including model reference and subsystems

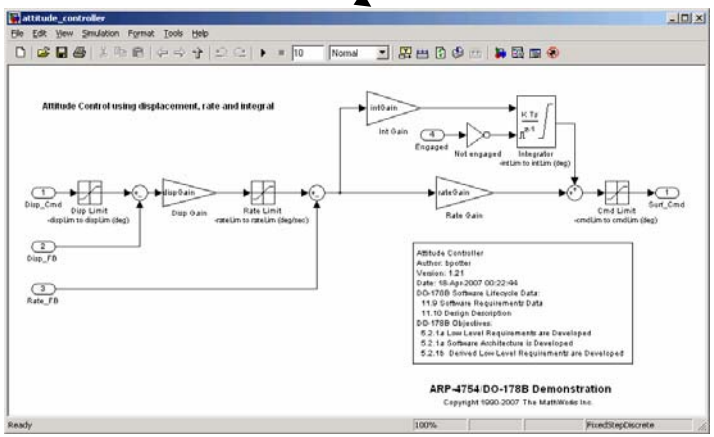
Top Model



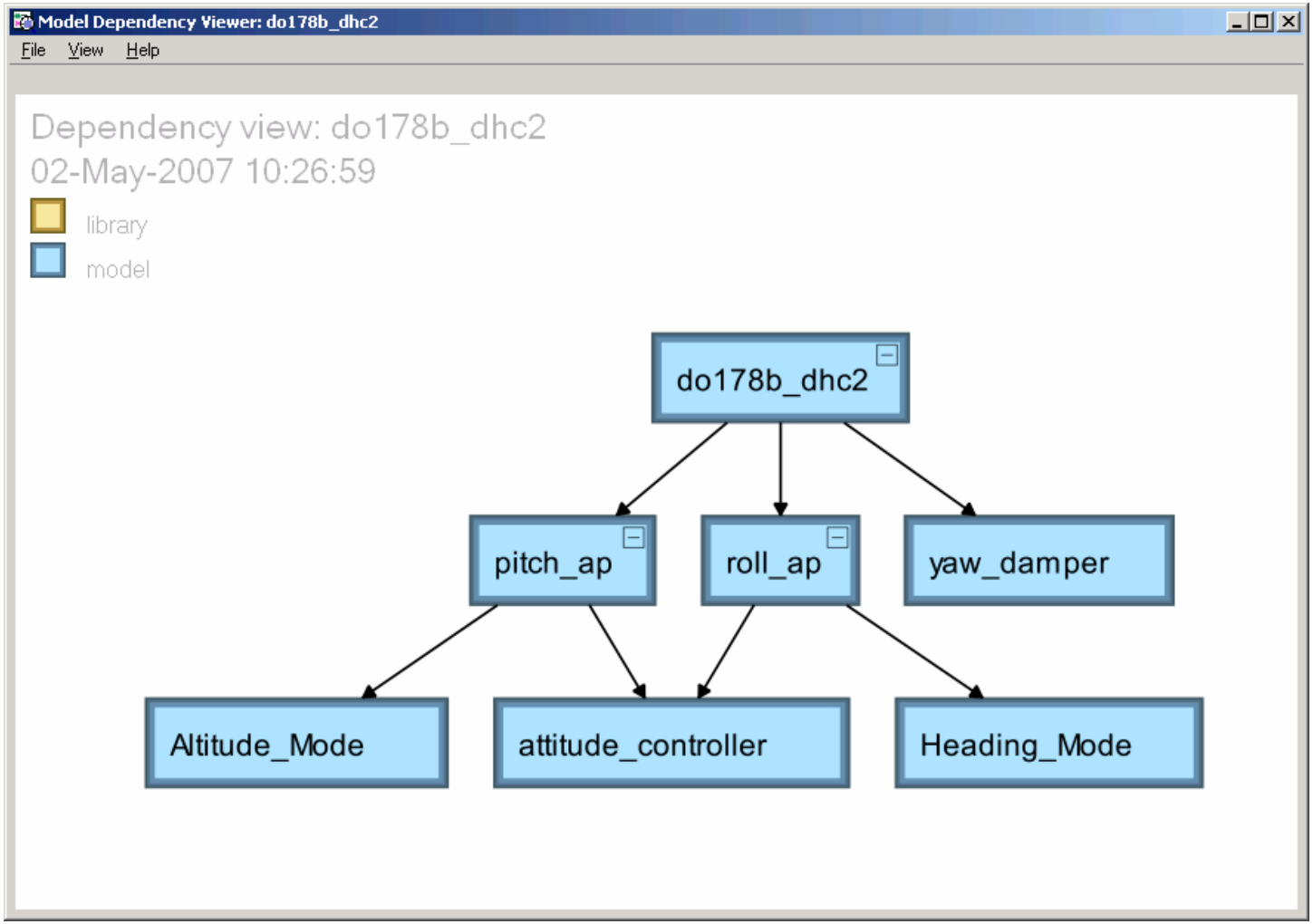
Subsystem



Reference Model



Model dependency viewer



Example Model Advisor report

Model Advisor - attitude_controller

File Edit View Help

Select All | Deselect All

Task Hierarchy: attitude_controller

- Model Advisor Task Manager
  - By Product
    - Simulink
    - Real-Time Workshop Embedded Code
    - Simulink Verification and Validation
    - DO-178B and MISRA-C Checks
      - Display software lifecycle data
      - Display model version information
      - Check safety related optimization s
      - Check safety related solver diagno
      - Check safety related sample time c
      - Check safety related data validity -
      - Check safety related data validity -
      - Check safety related data validity -
      - Check safety related data validity -
      - Check safety related type conversi
      - Check safety related connectivity -
      - Check safety related connectivity -
      - Check safety related connectivity -
      - Check safety related compatability
      - Check safety related model referer

By Product

Model Advisor

To process all enabled validations in this folder and generate a r
To display the report automatically after validations are processe
To display the last report generated, click the path link listed for '

Analysis

Run Selected Checks

☑ Show report after run

Last Report

From node: By Product

Report: C:\local_work_area\demos\autopilot_R2007a_m

Date/Time: 02-May-2007 10:35:27

Summary: 42 3

Model Advisor -- C:\local_work_area\demos\autopilot_R2007a_mdlref\slprj\model...

Model Advisor Report for 'attitude_controller'

Model version: 1.21

Generated on: 02-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 Inport 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

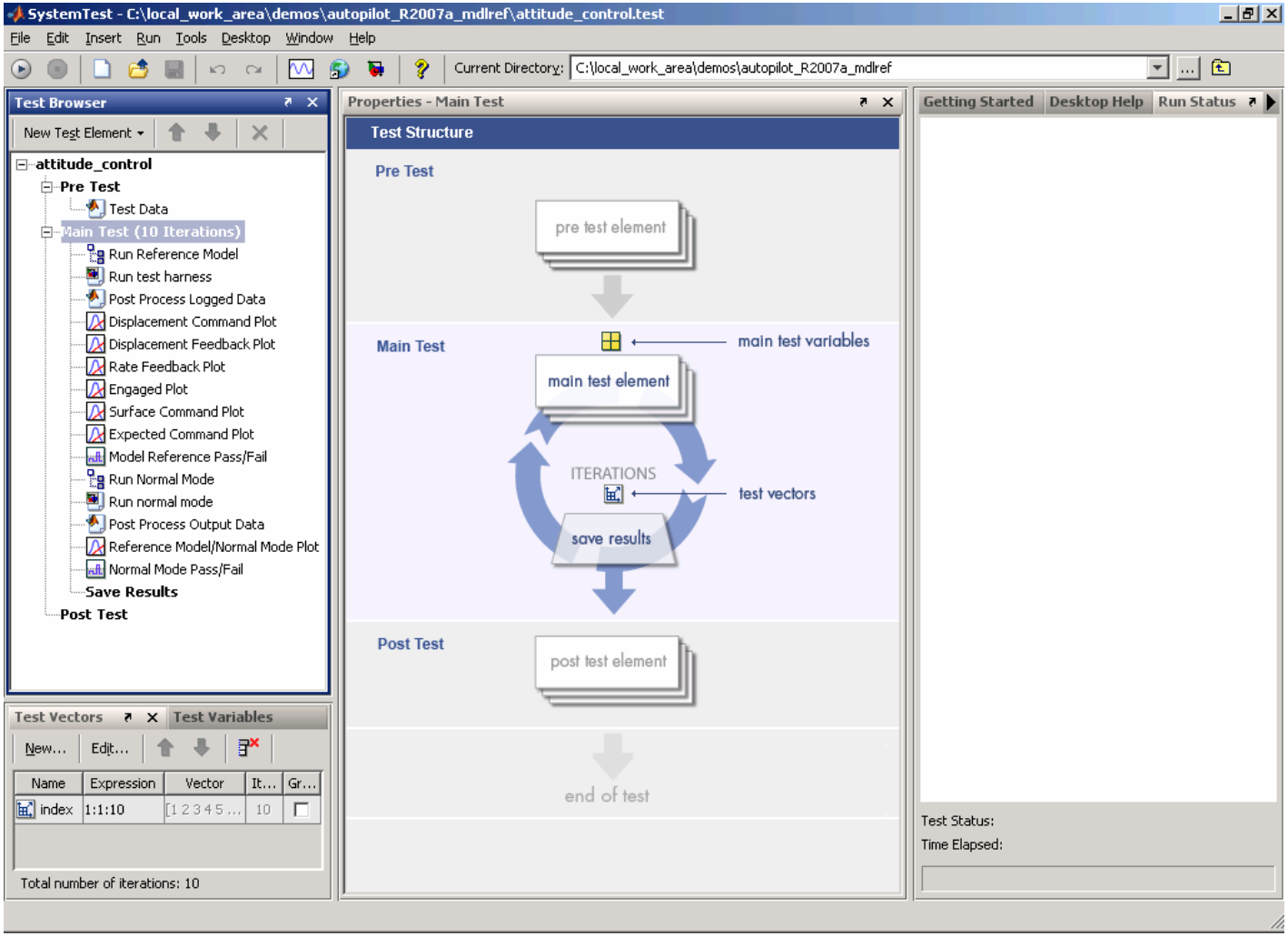
Check for blocks not supported by Real-Time Workshop:

Done

Design Verification for Model-Based Design

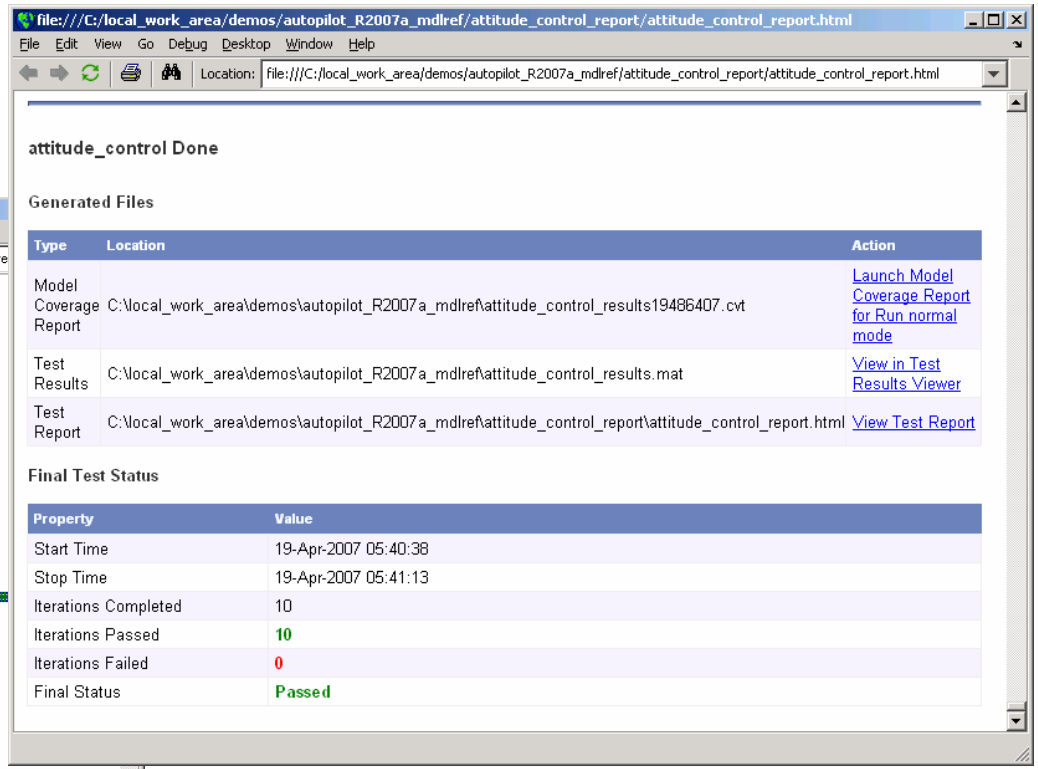
- Requirements based test cases
  - Automated testing using SystemTest/Simulink V&V
  - Traceability using Requirements Management Interface
  - Capability to inject faults for FMEA
- Robustness testing and analysis
  - Built in Simulink run-time diagnostics
  - Formal proofs using Simulink Design Verifier
- Coverage Analysis
  - Verify structural coverage of model
  - Verify data coverage of model

SystemTest for requirements based testing



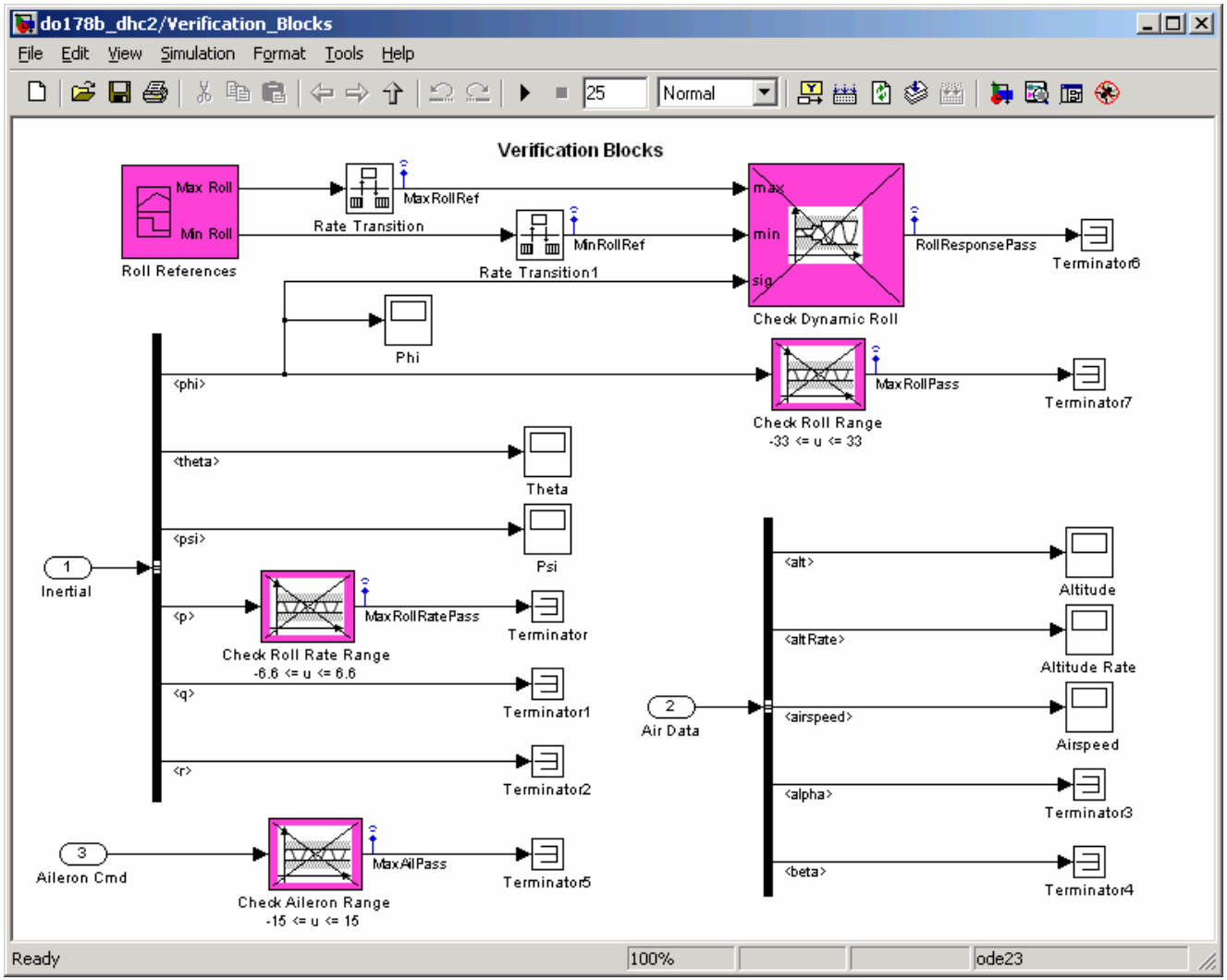
SystemTest – example report

Data Plotting and expected results comparisons



Summary of results

Signal Builder and Assertion Blocks



Model coverage report example – signal ranges

attitude_controller Coverage Report

File Edit View Go Debug Desktop Window Help

Location: file:///C:/local_work_area/demos/autopilot_R2007a_mdlref/attitude_control_results19486407.html

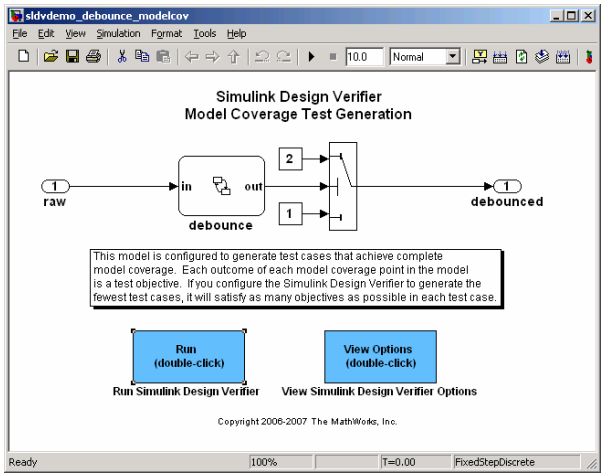
Signal Ranges:

Hierarchy	Test 1		Test 2		Test 3		Test 4		Test 5		Test 6		Test 7		Test 8		Test 9		Test 10		Overall	
	Min	Max	Min	Max	Min	Max	Min	Max	Min	Max	Min	Max	Min	Max	Min	Max	Min	Max	Min	Max	Min	Max
attitude_controller																						
. . . Integrator	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	-3	3	-3	3
. . . Not engaged	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
. . . Cmd Limit	-1	1	-9	9	-10	10	-1	1	-4	4	-5	5	-1	1	-7	7	-7.5	7.5	-4	4	-10	10
. . . Disp Limit	-1	1	-9	9	-10	10	0	0	0	0	0	0	0	0	0	0	0	0	-1	1	-10	10
. . . Rate Limit	-1	1	-9	9	-10	10	-1	1	-4	4	-5	5	0	0	0	0	0	0	-1	1	-10	10
. . . Disp Gain	-1	1	-9	9	-10	10	-1	1	-4	4	-6	6	0	0	0	0	0	0	-1	1	-10	10
. . . Int Gain	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	-2	2	-2	2
. . . Rate Gain	-1	1	-9	9	-10	10	-1	1	-4	4	-5	5	-1	1	-7	7	-8	8	-1	1	-10	10
. . . Sum	-1	1	-9	9	-10	10	-1	1	-1	1	-1	1	0	0	0	0	0	0	-1	1	-10	10
. . . Sum1	-1	1	-9	9	-10	10	-1	1	-4	4	-5	5	-1	1	-1	1	-1	1	-1	1	-10	10
. . . Sum2	-1	1	-9	9	-10	10	-1	1	-4	4	-5	5	-1	1	-7	7	-8	8	-4	4	-10	10

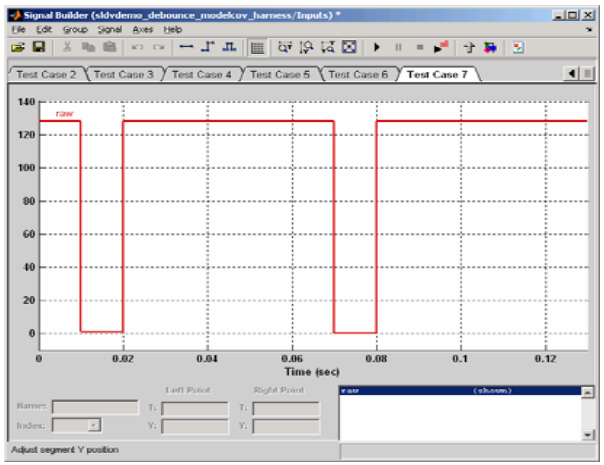
Done

Simulink® Design Verifier – Coverage Test

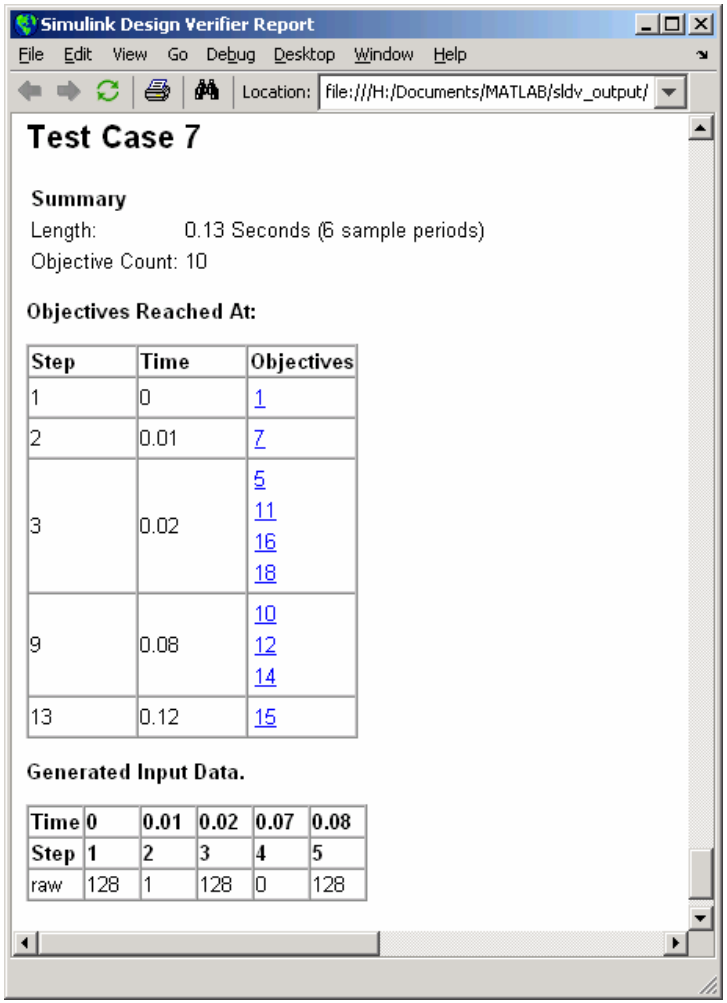
Model



Generated Test Cases



Test Report



Test Case 7

Summary

Length: 0.13 Seconds (6 sample periods)

Objective Count: 10

Objectives Reached At:

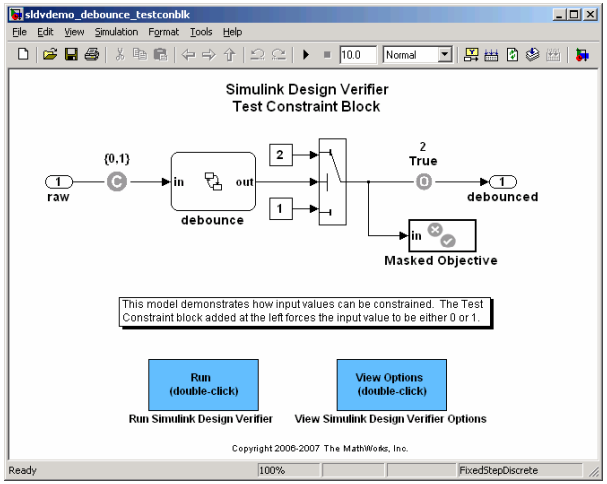
Step	Time	Objectives
1	0	1
2	0.01	7
3	0.02	5 11 16 18
9	0.08	10 12 14
13	0.12	15

Generated Input Data.

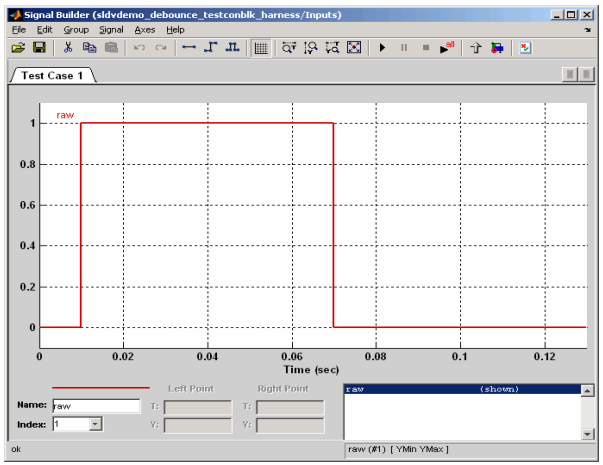
Time	0	0.01	0.02	0.07	0.08
Step	1	2	3	4	5
raw	128	1	128	0	128

Simulink Design Verifier – Objective Test

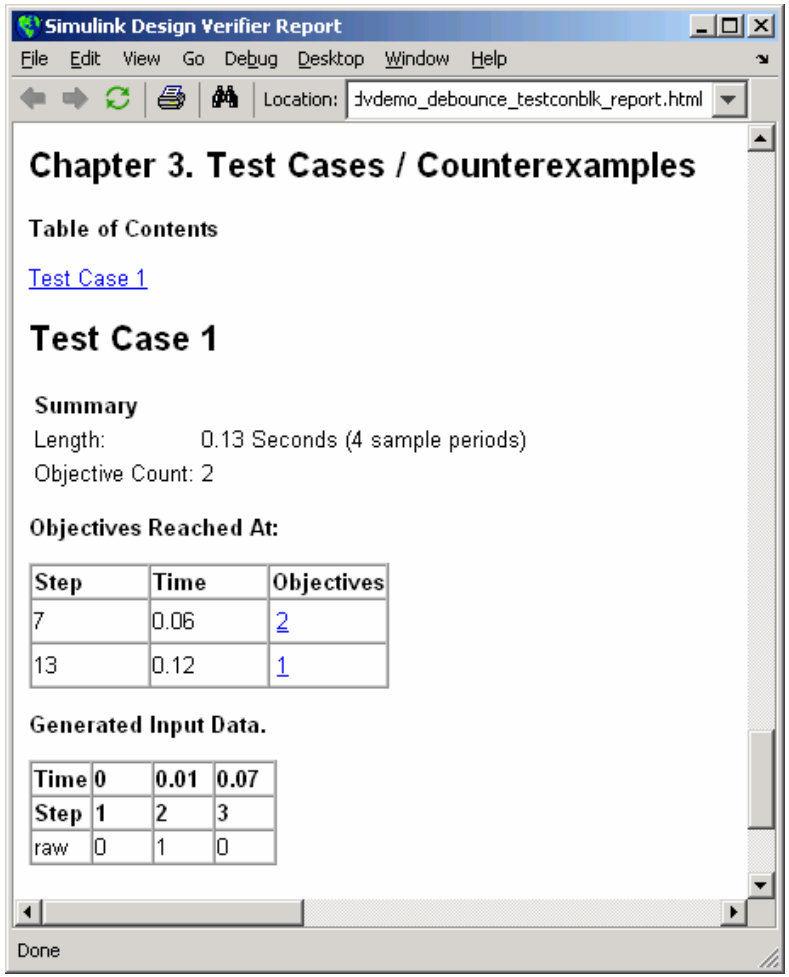
Model with Constraints and Objectives



Generated Test Cases

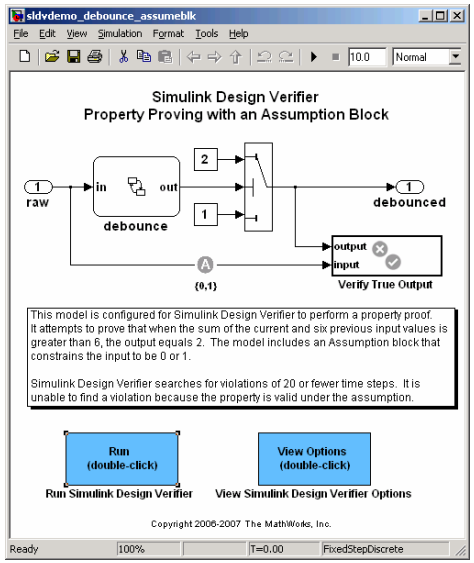


Test Report

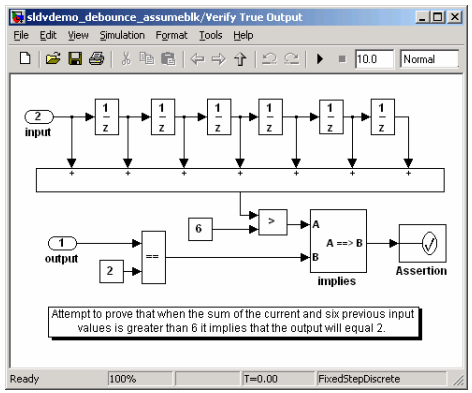


Simulink Design Verifier – Property Proving

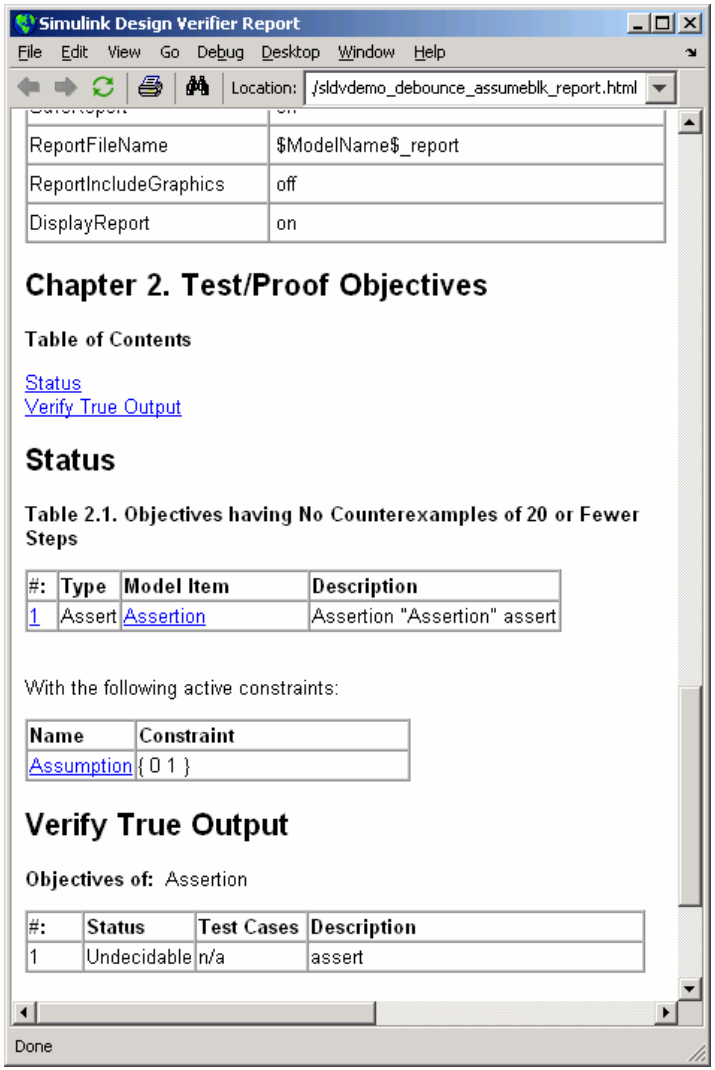
Model with Assumption and Objective



Property to be proven



Report



Design Process take-aways

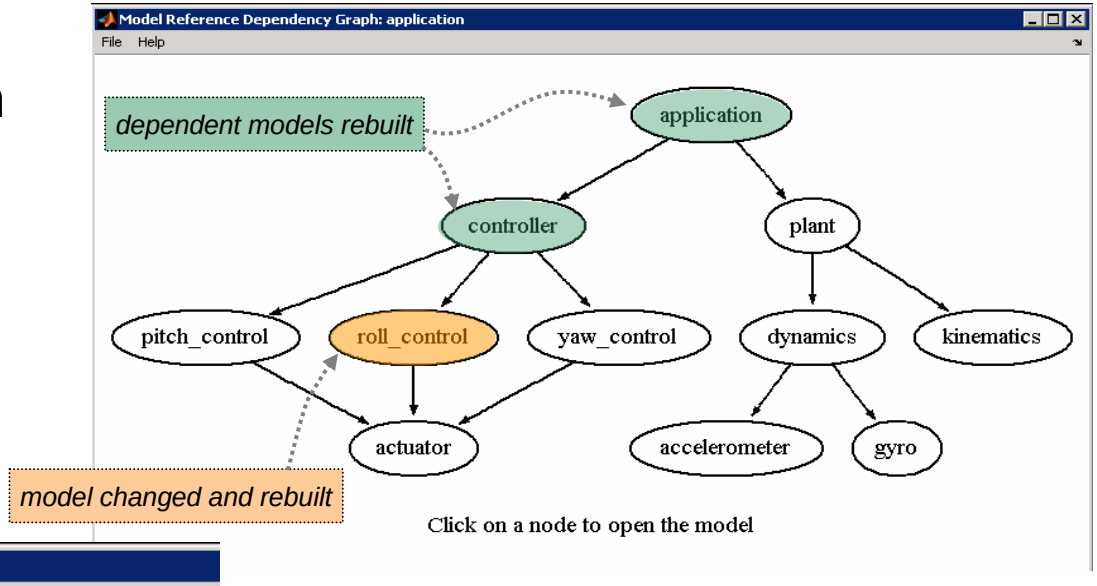
- Modular reusable implementations
  - Platform independent design and code
  - Scalable to large teams
- Consistent and compliant implementations
  - Common design language
  - Automated verification of standards compliance
- Efficient verification process
  - Develop verification procedures in parallel with design
  - Automated analysis techniques
  - Coverage analysis early in the process

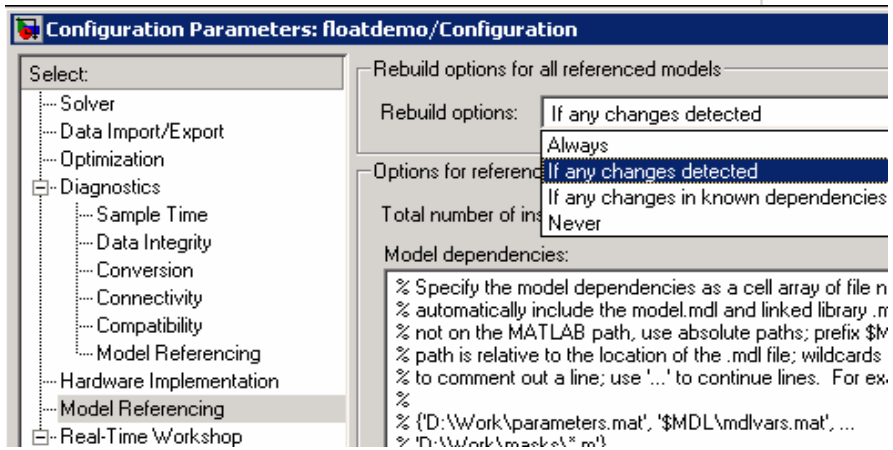
Coding Process for Model-Based Design

- 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

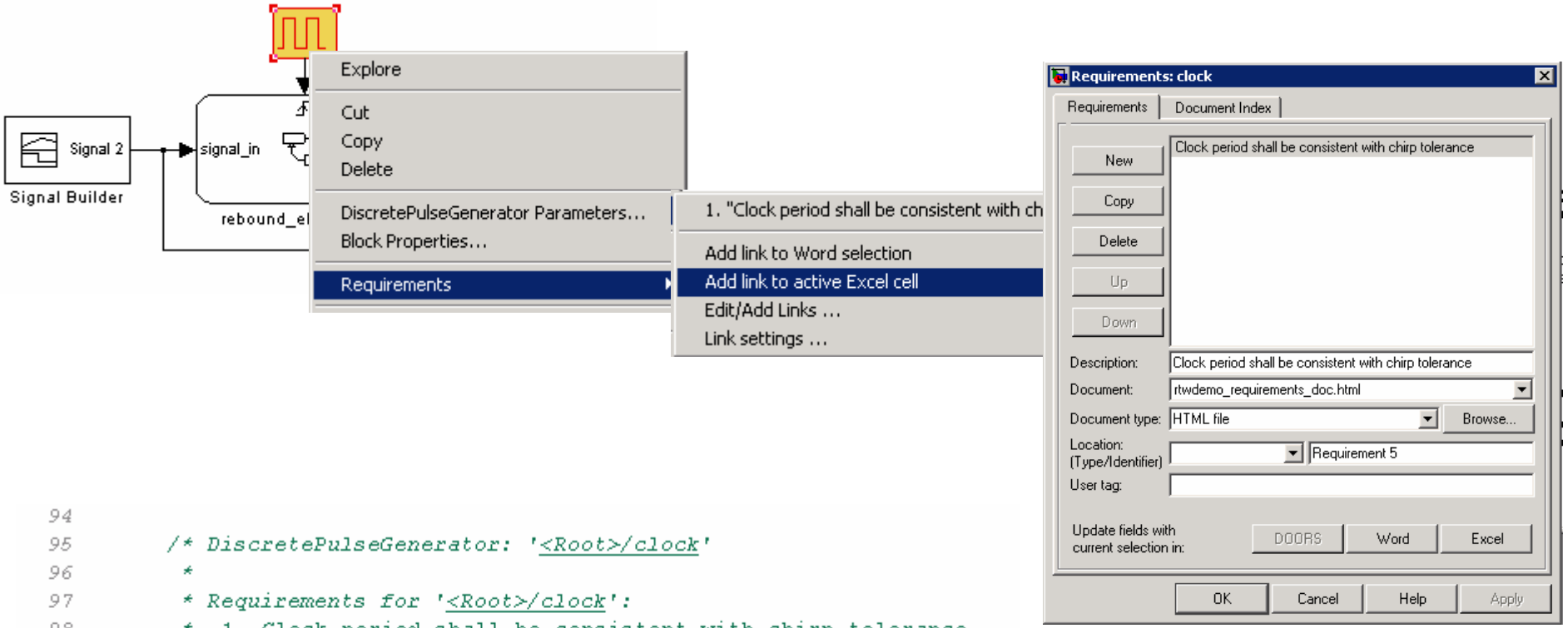
- Incremental code generation is supported via Model Reference
- When a model is changed, only models depending on it are subject to regeneration of their code





- Reduces application build times and ensure stability of a project's code
- Degree of dependency checking is configurable

Add Links to Requirements

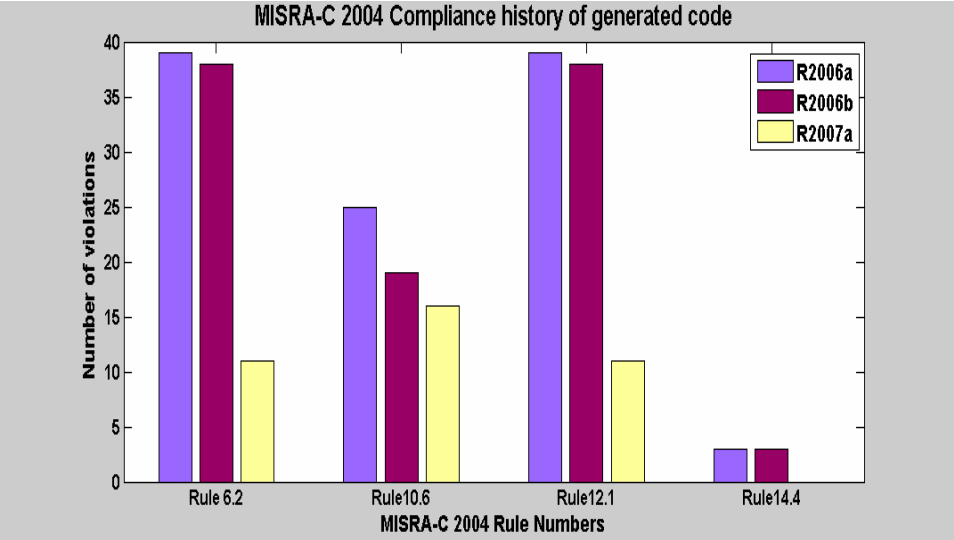



```
94
95      /* DiscretePulseGenerator: '<Root>/clock'
96       *
97       * Requirements for '<Root>/clock':
98       *  1. Clock period shall be consistent with chirp tolerance
99       */
100     rtb_clock =
101       (rtDWork.clockTickCounter < 1.0 &&
102        rtDWork.clockTickCounter >= 0) ?
103       1.0 :
104       0.0;
105     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

Coding Process take-aways

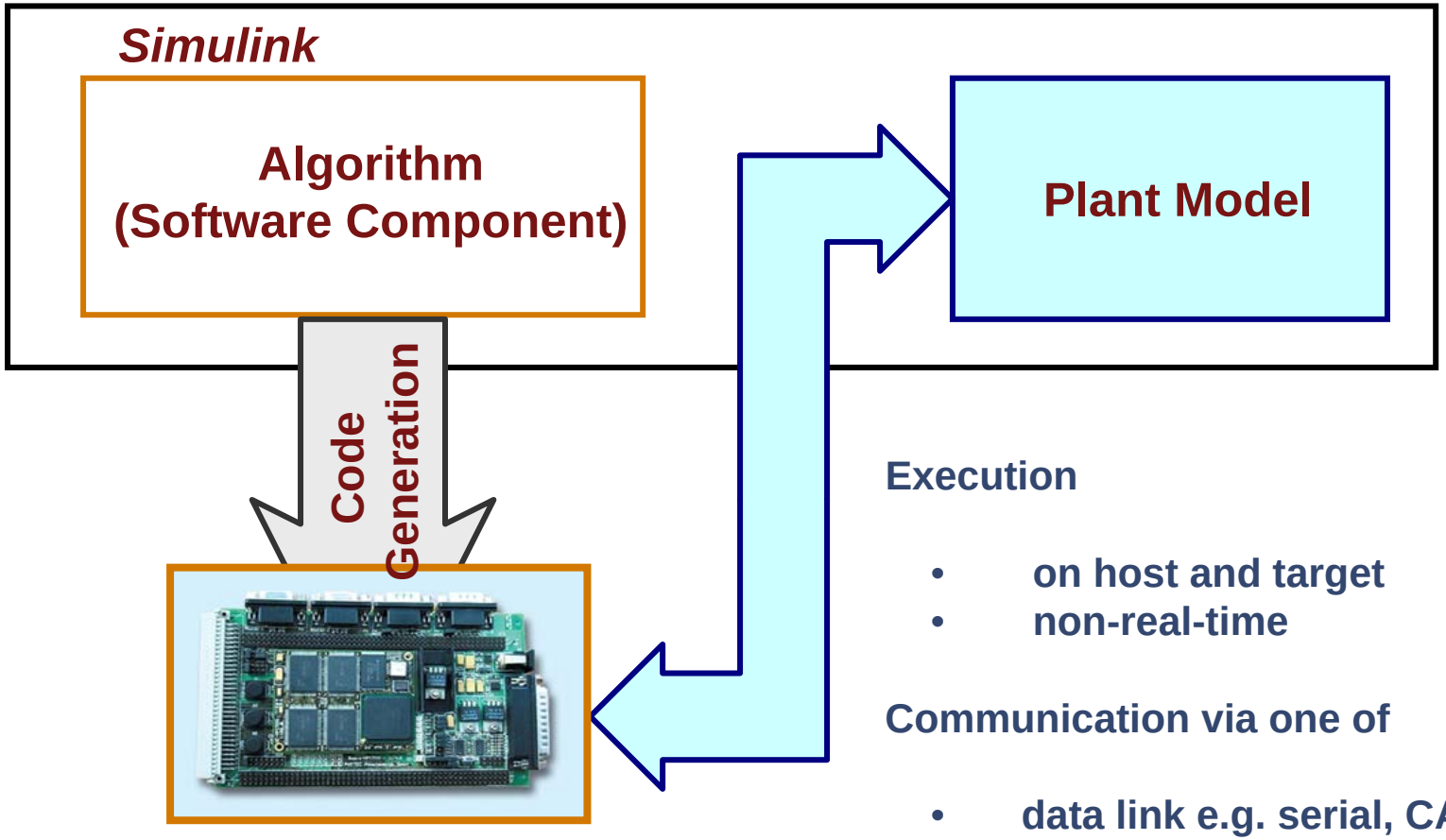
- Reusable and efficient source code
- Traceability
- MISRA-C compliance
- Static verification and analysis

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