The MathWorks MATLAB® & SIMULINK®

New Concepts and Tools for Effective Verification and Validation Based on Model Analysis

Master Class

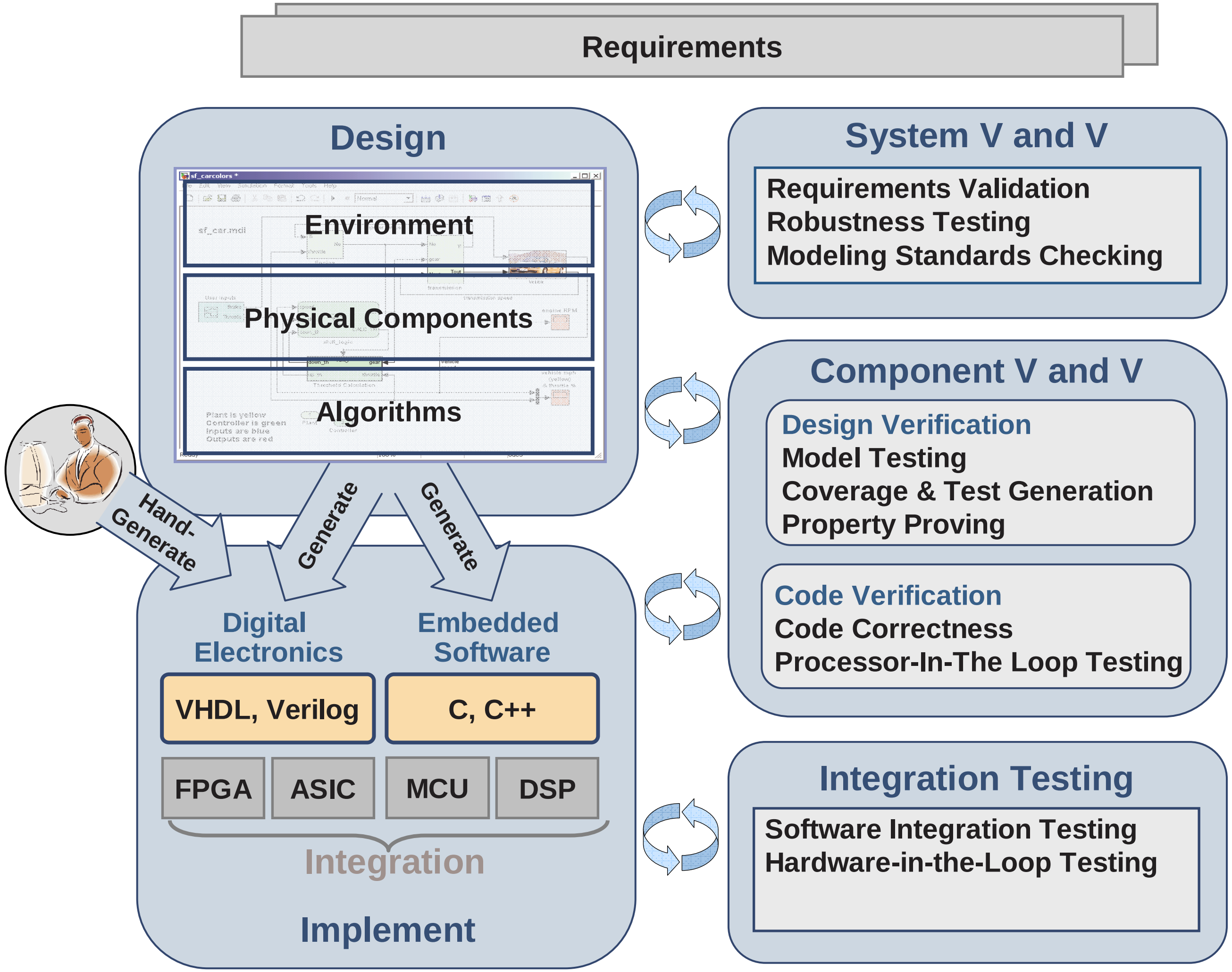
Today's Agenda

- Quick Demo
- Challenges
- Methods for Early Verification and Validation
  - Robustness Testing
  - Automatic Test Generation
  - Property Proving
- Questions and Answers

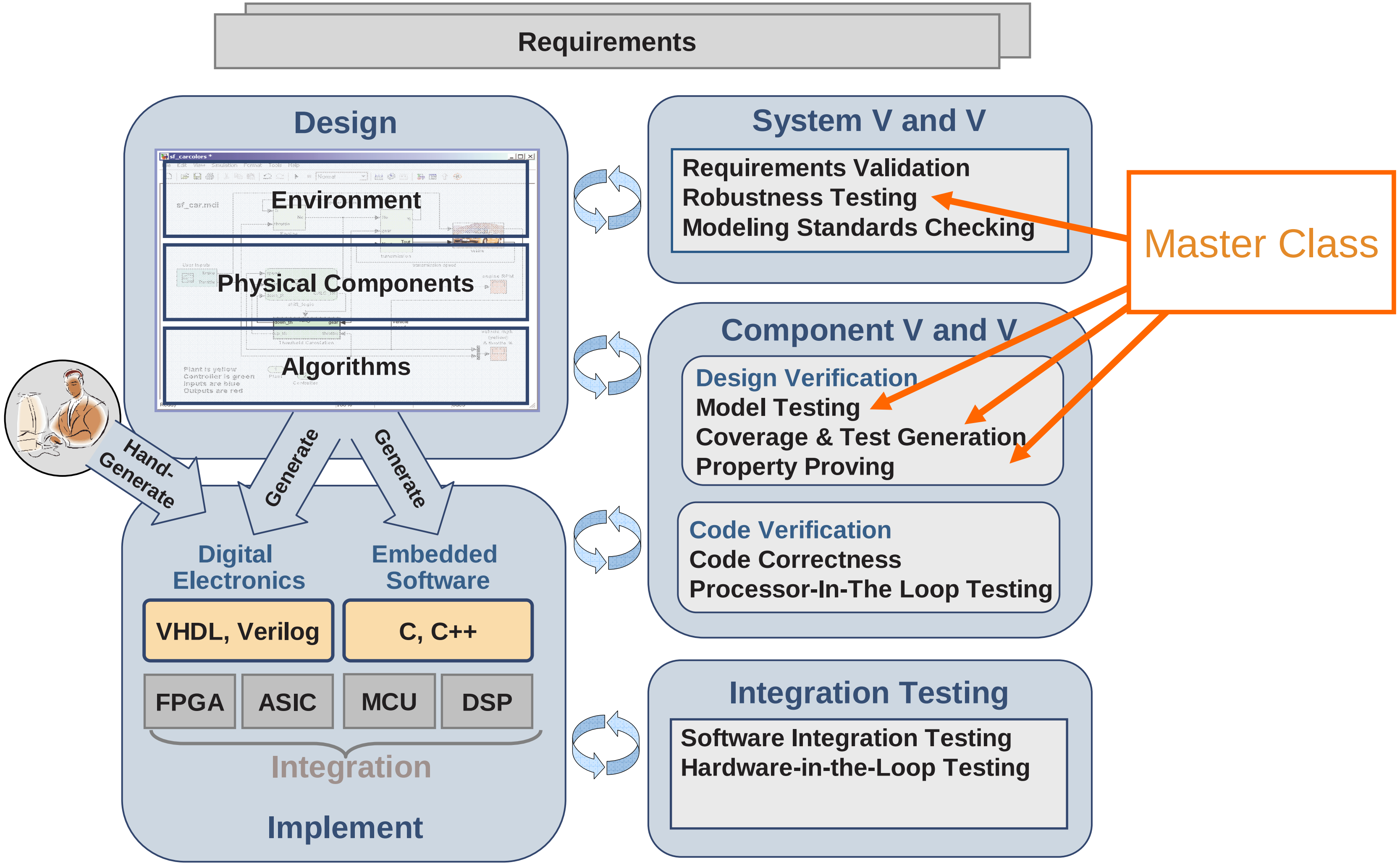
Poll

- Do you test your models?
- Do you have coverage requirements?
  - How hard is it to reach 100% coverage?

Address the Entire Development Process



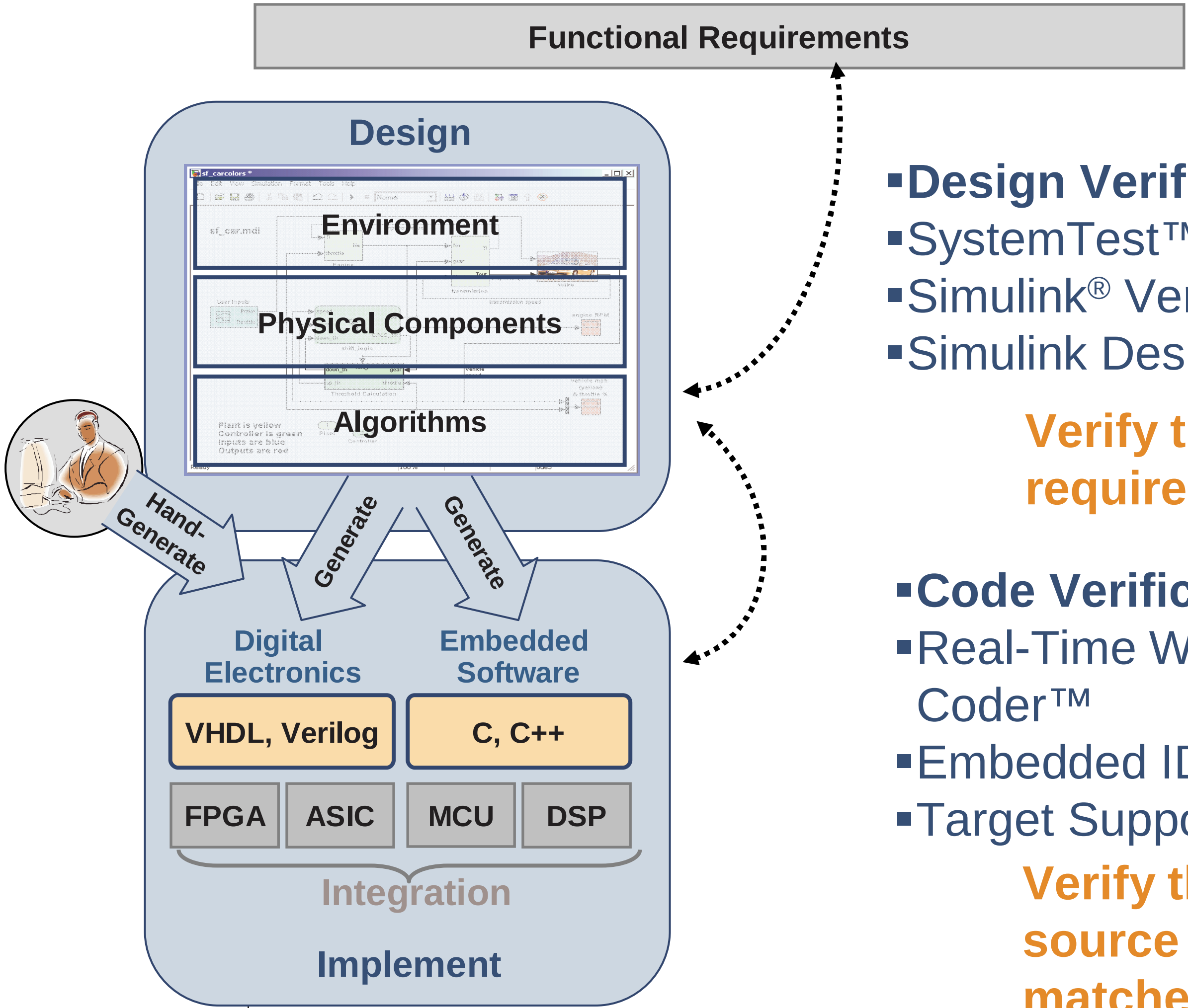
Address the Entire Development Process



Verification and Validation Challenges

- Management of tests and test assets
- Writing tests for 100% coverage of control logic is hard
- Some requirements are difficult to test

Testing in Simulation



- **Design Verification**
- SystemTest™
- Simulink® Verification and Validation™
- Simulink Design Verifier™

Verify that design meets requirements

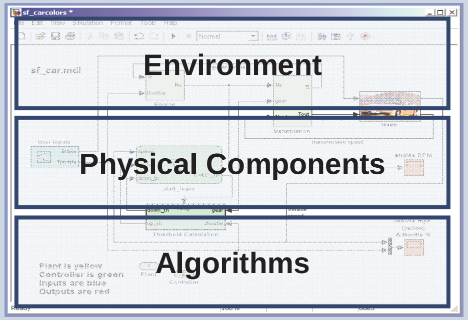
- **Code Verification**
- Real-Time Workshop® Embedded Coder™
- Embedded IDE Link™ products
- Target Support Package™ products

Verify that the behavior of source code and object code matches the model

Early Validation and Robustness Testing

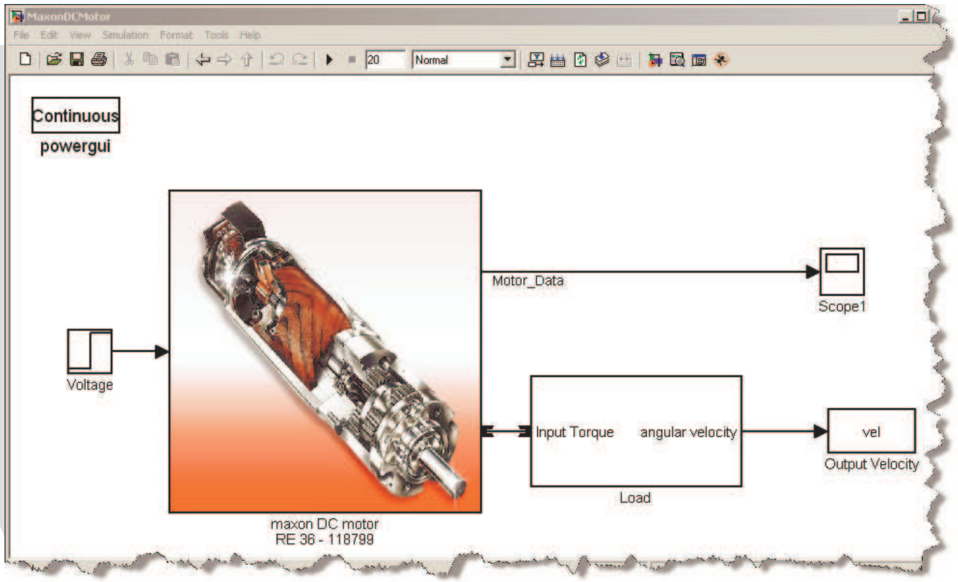
Requirements

Design



System V and V

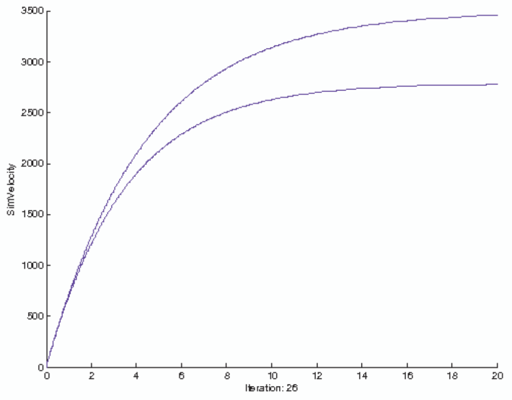
Requirements Validation
Robustness Testing
Modeling Standards Checking



System V and V - Example

- Evaluation of robustness of a DC Motor model
- Assessment of model accuracy in predicting performance variability of real systems

Plot Velocity Profiles



Assess Model Accuracy for Performance Variability

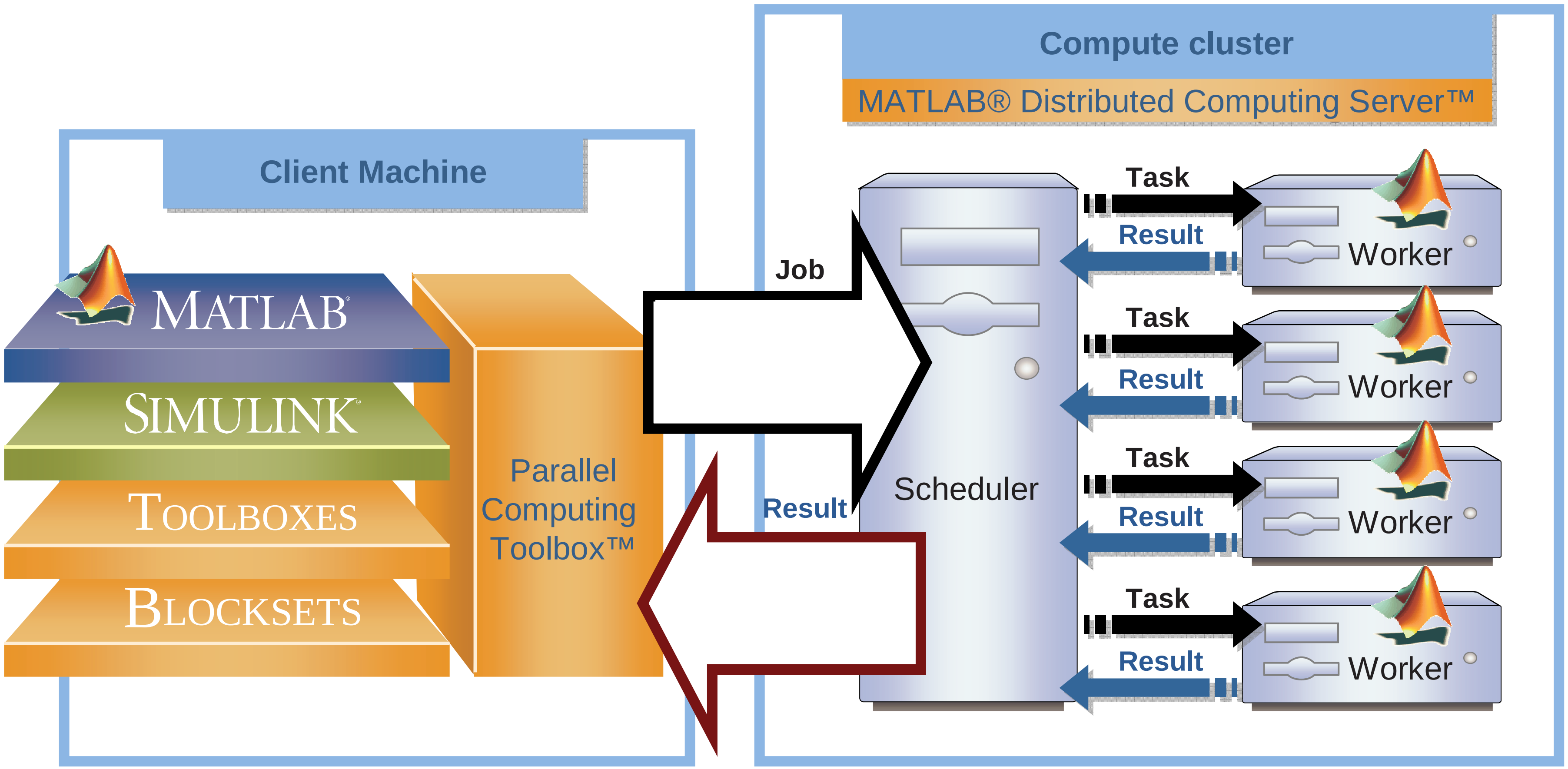
Test Variable	Expected Value	Tolerance Type	Tolerance Limit	Evaluates To
SimRiseTime 9.2257	expRiseTime 8.5222	Relative	tolerance 0.07	FALSE
SimSSVelocity 3455.9	expSSVelocity 3417.7	Relative	tolerance 0.07	TRUE

Saved Results

Name	Value
SimVelocity	<36541x1 double>
SimTime	<36541x1 double>
SimRiseTime	9.2257
SimSSVelocity	3455.9
expRiseTime	8.5222
expSSVelocity	3417.7
toleranceResult	0

Iteration 26 Failed

System Test with Distributed Computing



Management of Tests and Test Assets

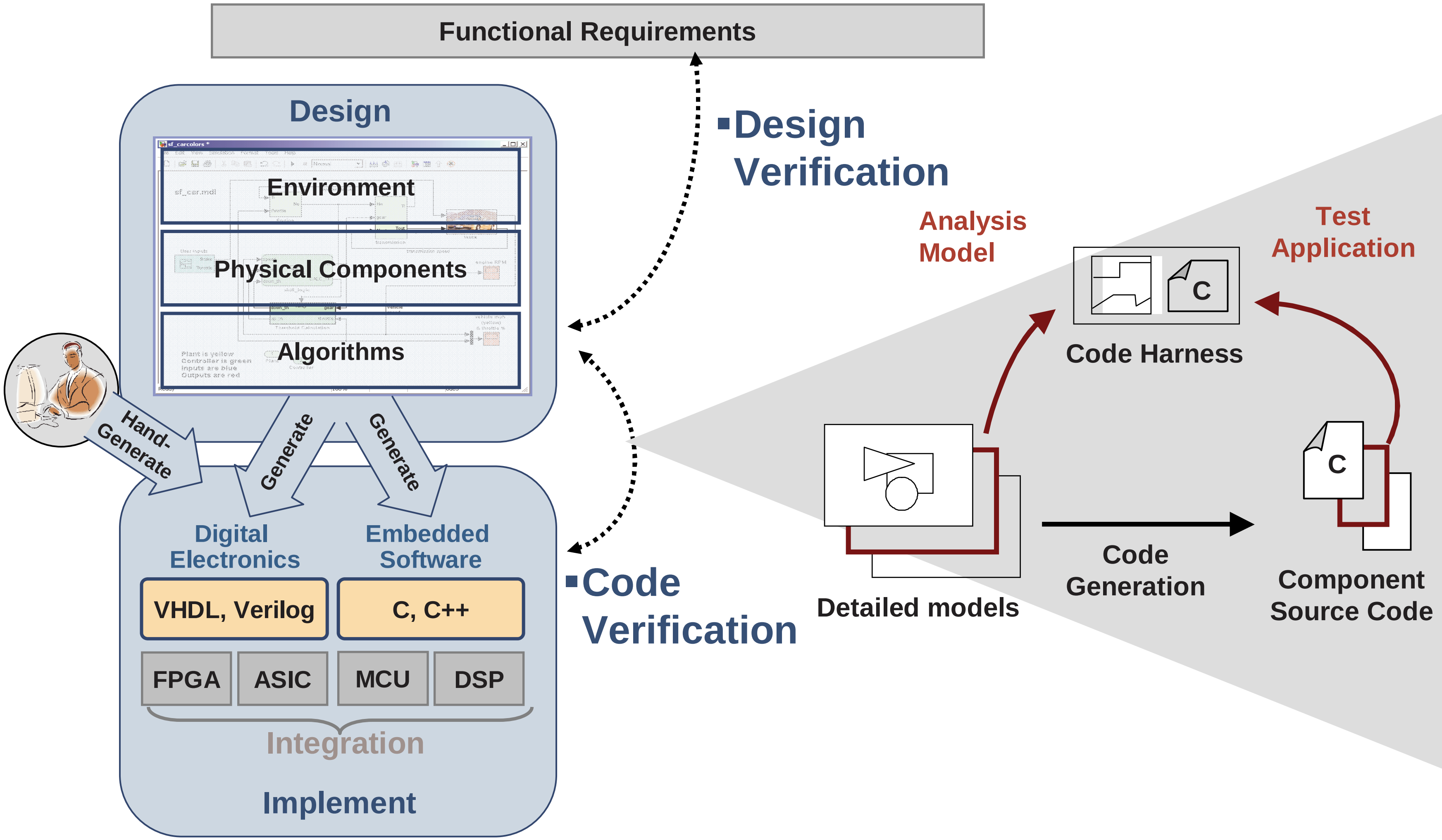
SystemTest™

- Authoring
  - Creating tests from requirements
  - Importing existing test data from Excel
  - Generating tests with Simulink Design Verifier

- Execution and Reporting
  - SystemTest plots and test report

- **Benefits**
  - Automate test execution
  - Build consistent test execution environment for repeatable results
  - Create baselines of design behavior and run them in regression
  - Continuously improve quality of models and generated code
  - Export tests and test results for testing on hardware

Test Generation Workflow



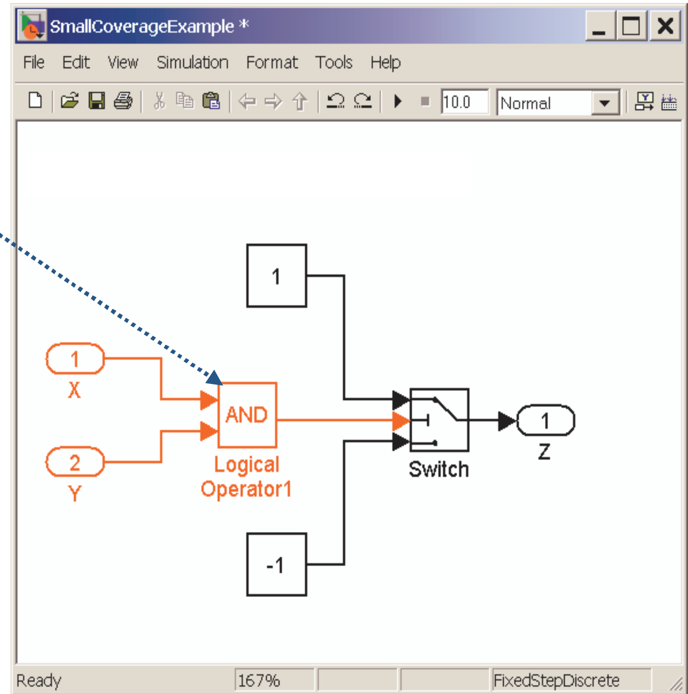
Model Coverage

Simulink Verification and Validation

- Structural metric
- Measure of test completeness

MC/DC Coverage
→ each condition independently changes the decision outcome

```
TT, FT
TT, TF
```




```
if (X & Y)
  Z = 1;
else
  Z = -1;
end
```

Decision

Condition

Example MC/DC Coverage

Model Coverage Tool

Simulink Verification and Validation

- Model Coverage tool reports coverage metrics
- User must provide input data for the simulation

Subsystem "Logical Operator"

Parent: coverage_example_harness/Test Unit (copied from coverage_example)
Uncovered Links:

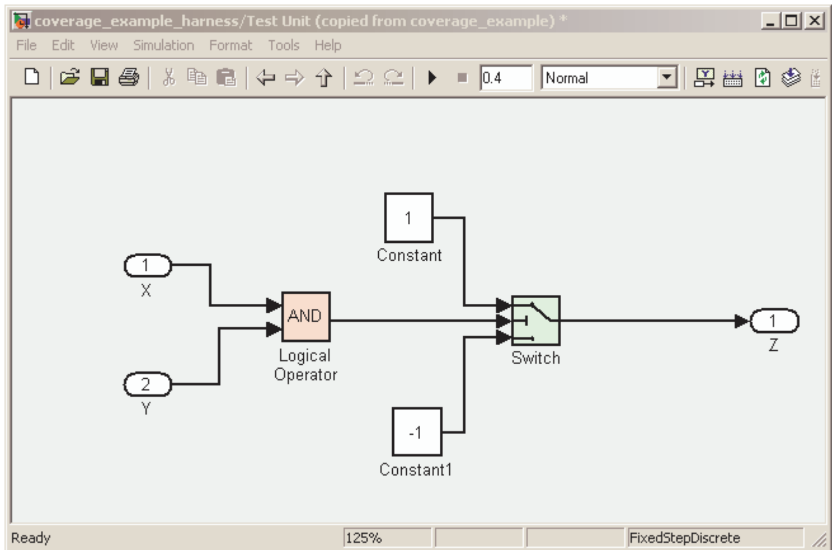
Metric	Coverage
Cyclomatic Complexity	0
Condition (C1)	100% (4/4) condition outcomes
MCDC (C1)	50% (1/2) conditions reversed the outcome

Conditions analyzed:

Description:	True	False
input port 1	6	3
input port 2	3	6

MC/DC analysis (combinations in parentheses did not occur)

Decision/Condition:	True Out	False Out
expression for output		
input port 1	TT	(FT)
input port 2	TT	TF



Objectives for Test Generation

Simulink Design Verifier

Affects (X & Y) to be T and F?

Affects (X & Y) to be T and F?

TT, FT
TT, TF

```
if (X & Y)
  Z = 1;
else
  Z = -1;
end
```

Chapter 2. Test Objectives

Table of Contents

Status

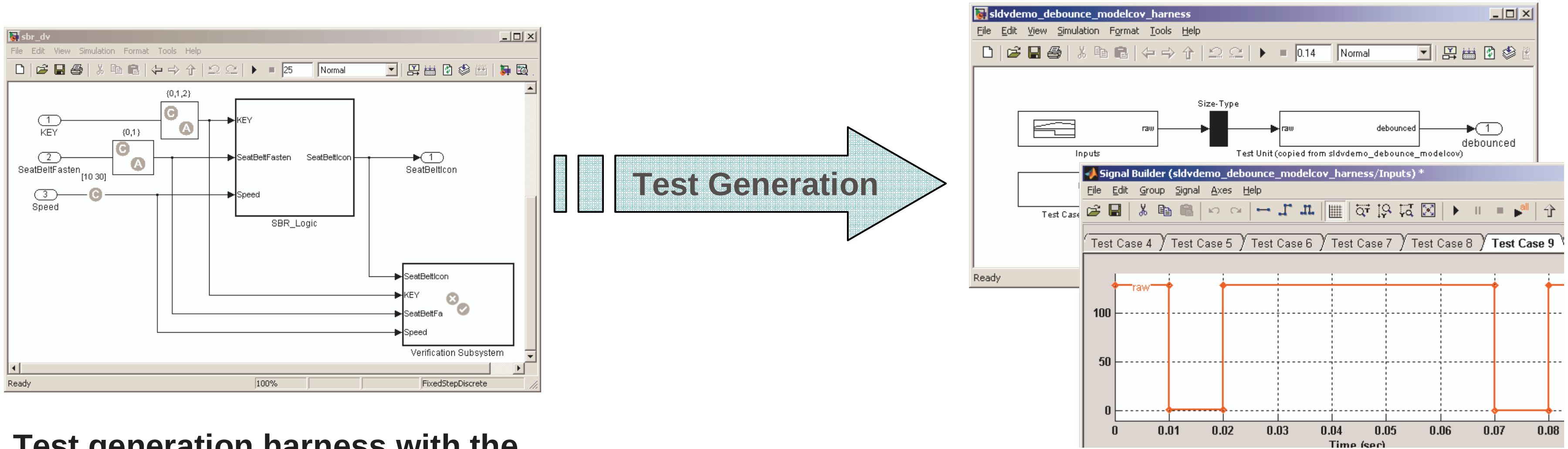
Table 2.1. Objectives Satisfied

#:	Type	Model Item	Description
1	Decision	Switch	Switch "Switch": logical trigger input false (output is from 3rd input port)
2	Decision	Switch	Switch "Switch": logical trigger input true (output is from 1st input port)
3	Condition	Logical Operator	Logic "Logical Operator": input port 1 T
4	Condition	Logical Operator	Logic "Logical Operator": input port 1 F
5	Condition	Logical Operator	Logic "Logical Operator": input port 2 T
6	Condition	Logical Operator	Logic "Logical Operator": input port 2 F
7	Mcdc	Logical Operator	Logic "Logical Operator", MCDC expression for output with input port 1 T
8	Mcdc	Logical Operator	Logic "Logical Operator", MCDC expression for output with input port 1 F
9	Mcdc	Logical Operator	Logic "Logical Operator", MCDC expression for output with input port 2 T
10	Mcdc	Logical Operator	Logic "Logical Operator", MCDC expression for output with input port 2 F

Test Generation for Coverage

Simulink Design Verifier

- Generating tests to reach coverage objectives

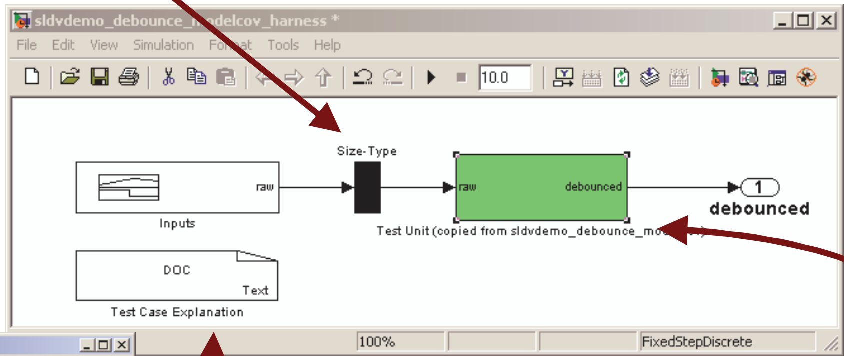


Test generation harness with the copy of the original model

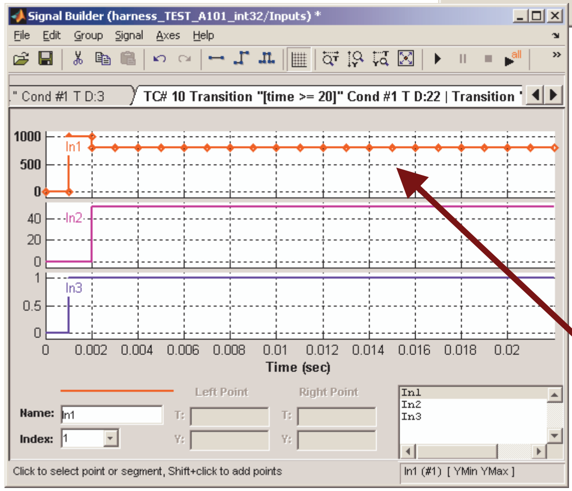
Test inputs that ensure complete coverage

Test Generation Results – Harness Model

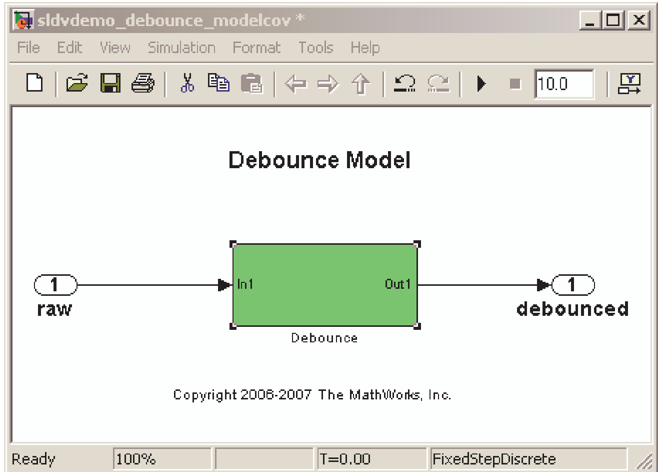
An interface block builds up vectors and cast signals to the needed data types



Original model copied to the harness



Test Cases are captured in a Signal Builder block

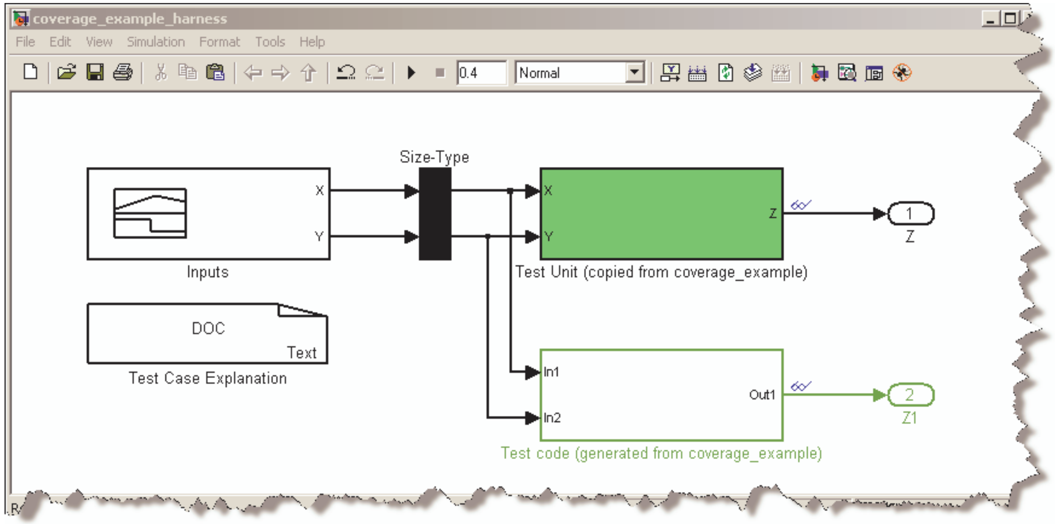


Input data sequences drive system from its initial configuration

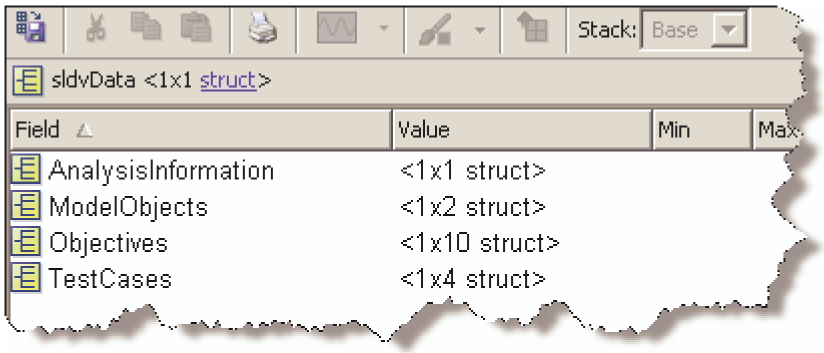
Code Testing with Generated Signals

Simulink

- Software-in-the-loop
  - On the host
- Processor-in-the-loop
  - On the target processor



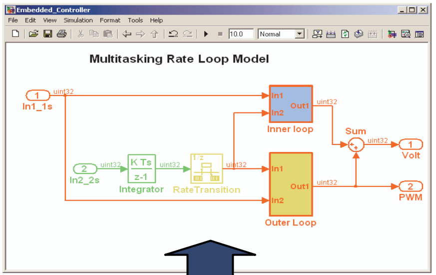
- Independent code testing environment
  - Generated signals and model outputs are saved as a .mat data file
  - Exported input signals drive code tests
  - Exported model outputs become expectation values for code testing



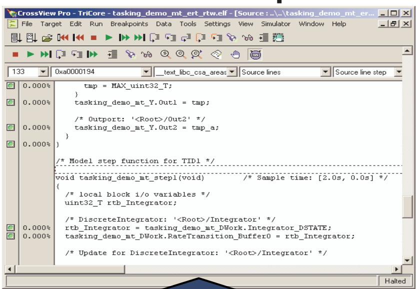
Processor-In-The-Loop Testing

Embedded IDE Link™ TS (for Altium® TASKING®)

Simulink:

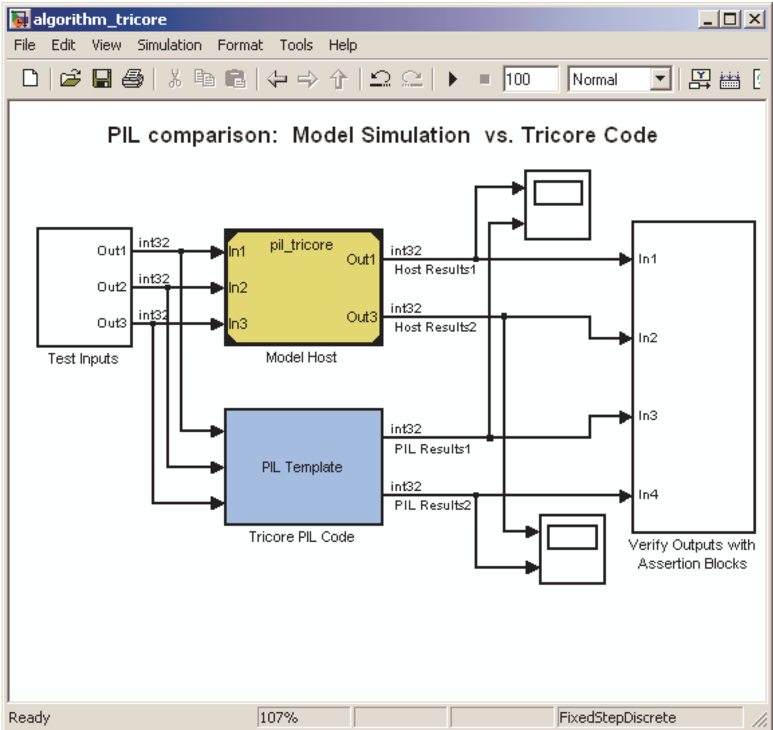


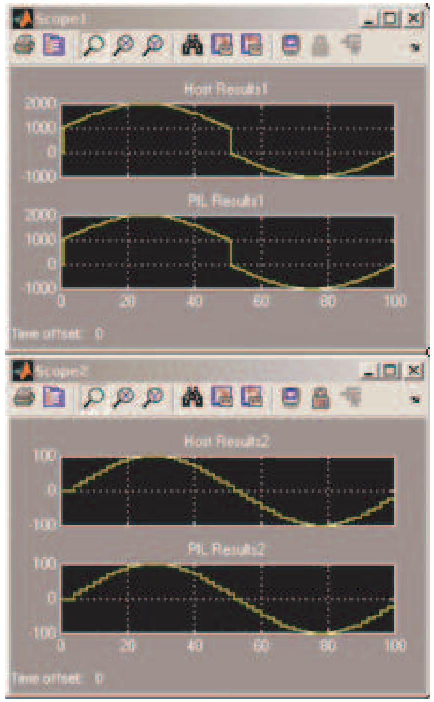
Real-Time Workshop® and TASKING:



ECU:

- Model in simulation and code on the processor running in parallel

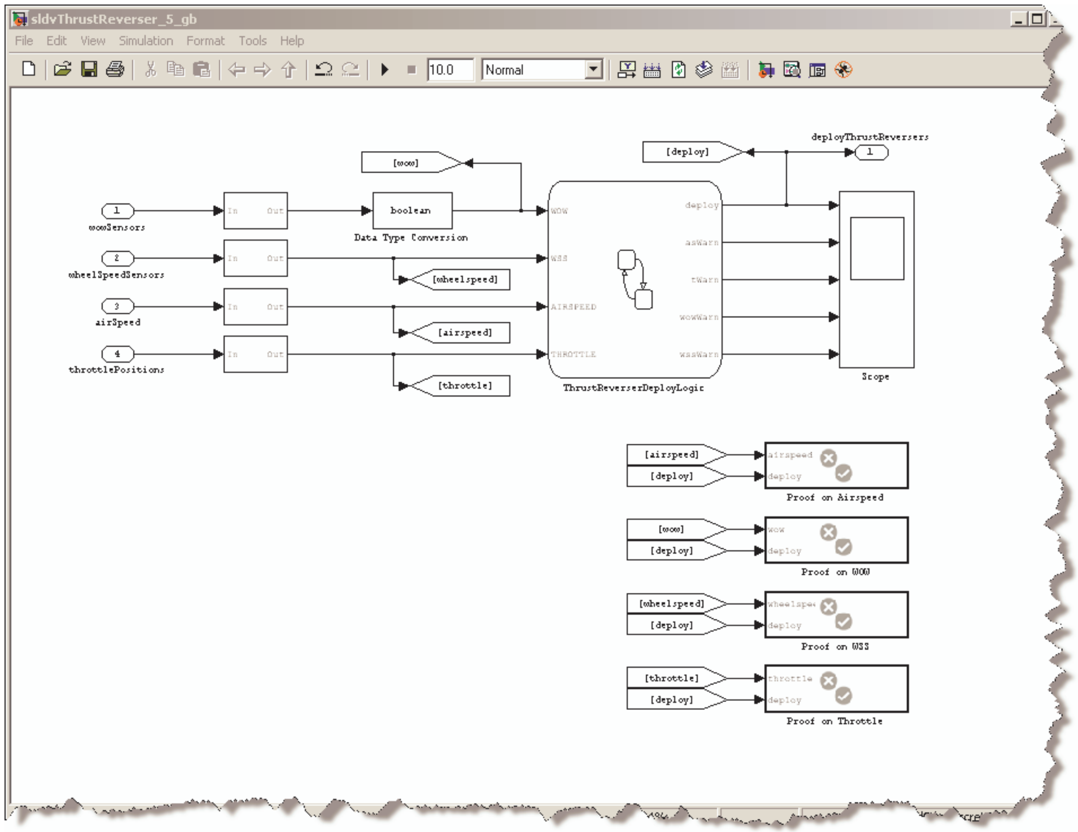




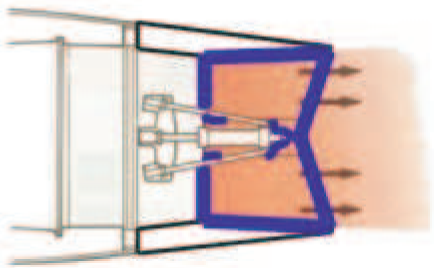
PIL also provides execution profiling, code coverage reports, and interactive debugging

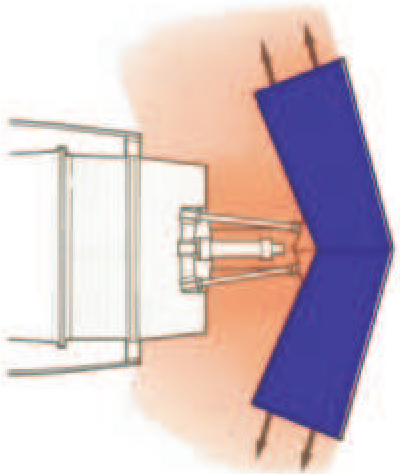
Demonstration

- Demonstration of test generation with Simulink Design Verifier



Thrust Reversers





Thrust Reversers Should not be Deployed During Flight

Lauda Air B767 Accident Report

SYNOPSIS

Prepared for the WWW by

Hiroshi Sogame
Safety Promotion Comt.
All Nippon Airways

U.S. Orders Thrust Reverser Deactivated on 767s

By Barry James Publishec

PARIS: The Federal Aviation Administration in Washington ordered U "deactivate" engine thrust reversers on Boeing 767 jetliners. Such a d... crash of an Austrian Lauda Air jet in Thailand nearly three months ago.

The aviation administration did not cite the in-flight deployment of one of the reversers as the cause of the accident. But it said it had established that a hydraulic failure could cause the devices to deploy in flight. Thrust reversers are designed to slow an aircraft after landing or an aborted takeoff.

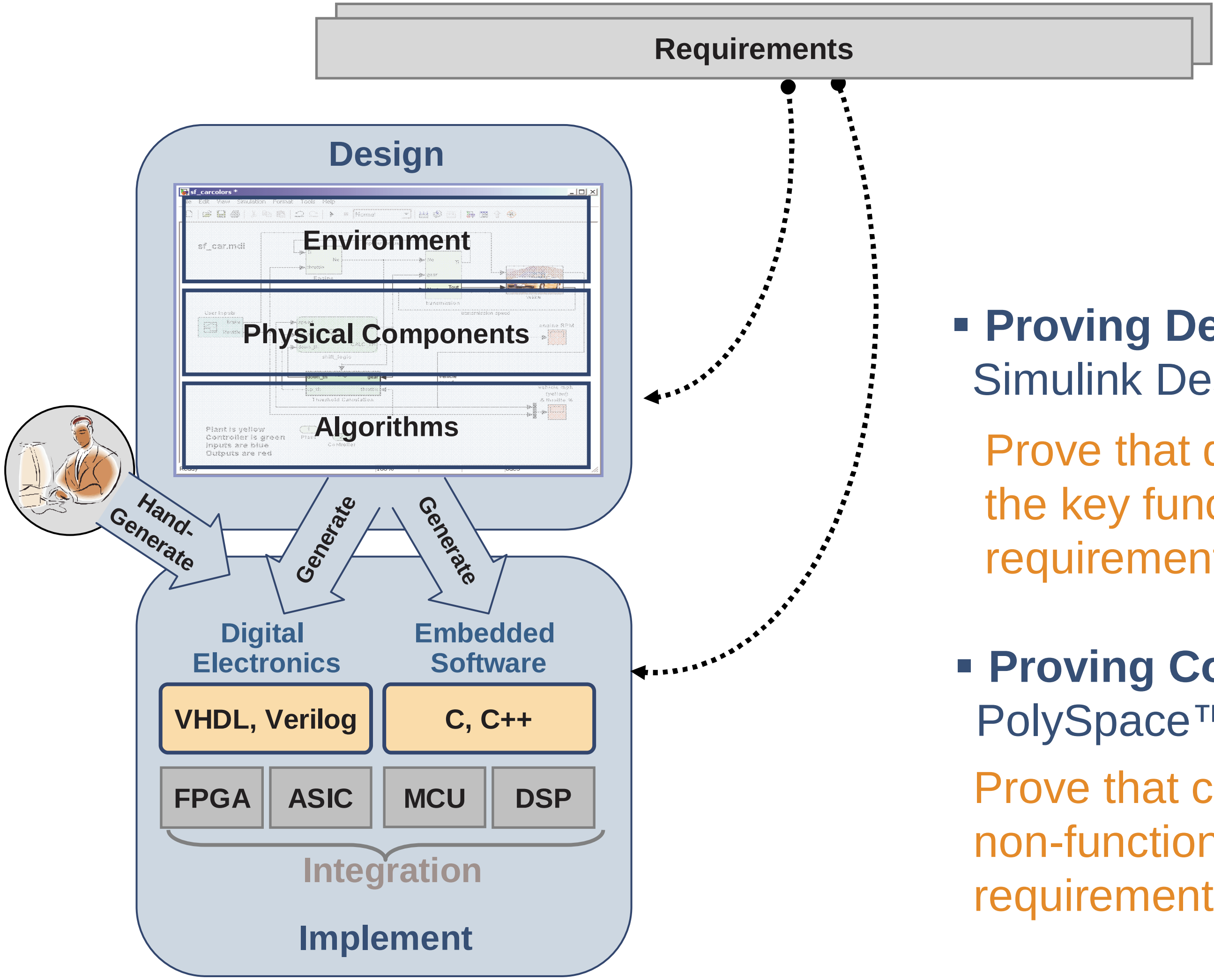
During the Lauda Air disaster on May 26, the pilot reported that a reverser had deployed in flight, sending most of the massive 56,000-pound thrust of one of the two Pratt & Whitney 4000 engines the wrong way.

All 223 people aboard were killed as the plane broke up in flight.

Thrust Reverser Deployment Requirements

- The following requirements shall be met prior to deploying the thrust reversers:
  - Weight on Wheels
    - Each main gear, each redundant
  - Wheel Speed Sensors
    - Each main gear
  - Airspeed Limit
    - Redundant Sensors
  - Throttle Positions
    - Each throttle, each redundant

Proving



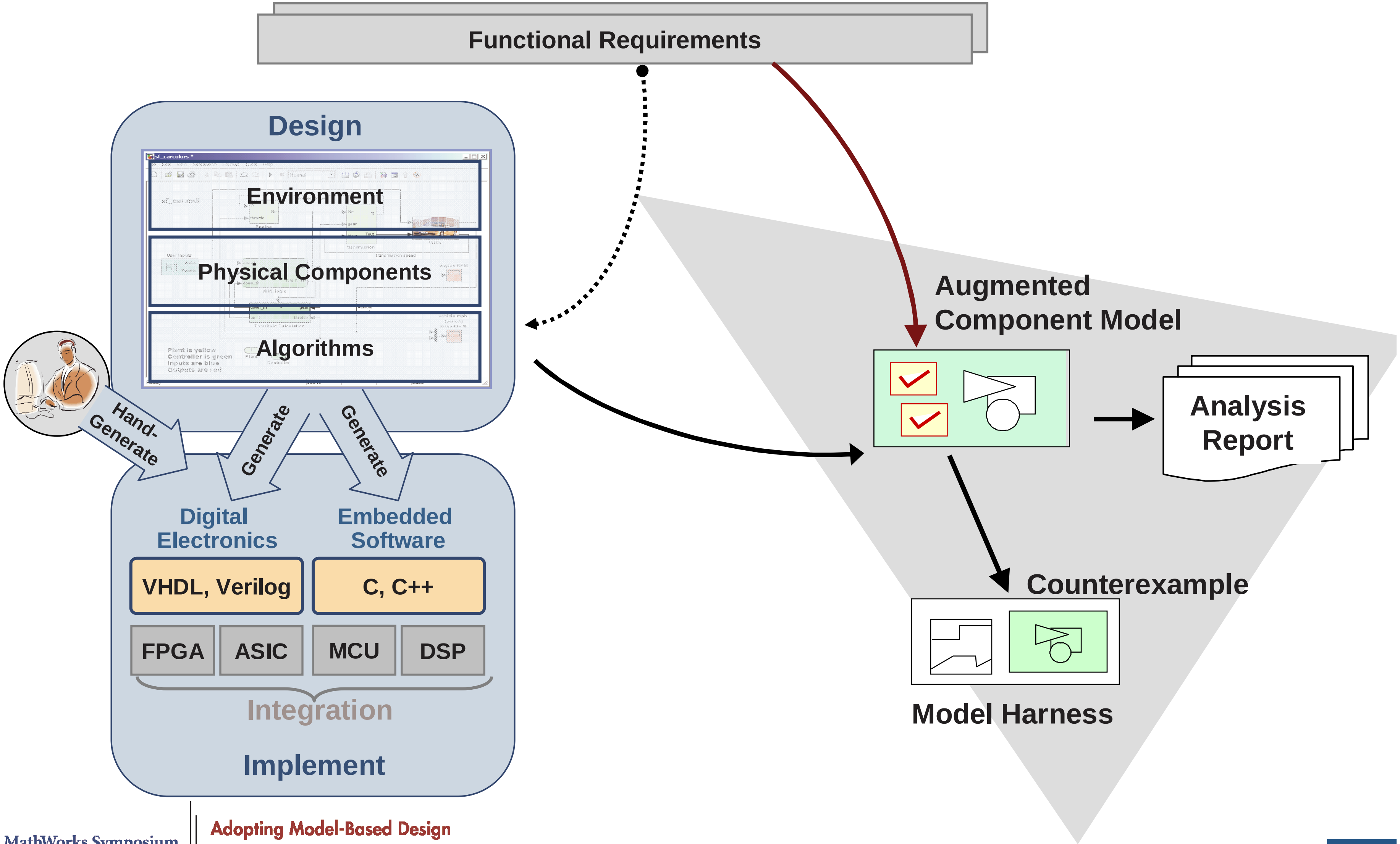
- **Proving Design Properties**
  Simulink Design Verifier

  Prove that design meets the key functional requirements

- **Proving Code Correctness**
  PolySpace™ Server for C/C++

  Prove that code meets non-functional runtime requirements

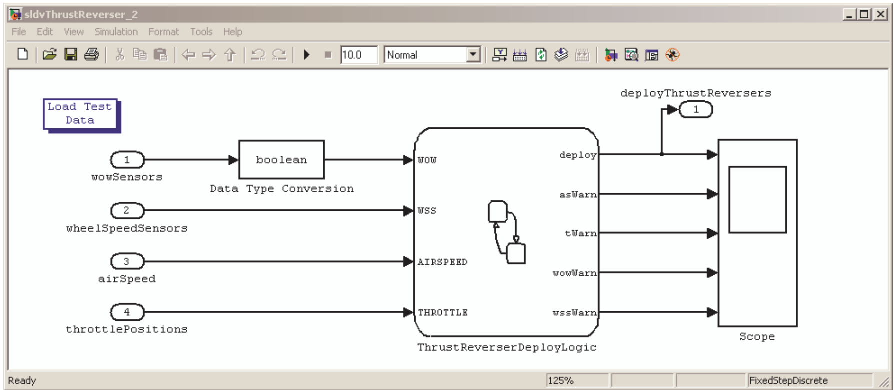
Property Proving Workflow



Property Proving – Overview

Simulink Design Verifier

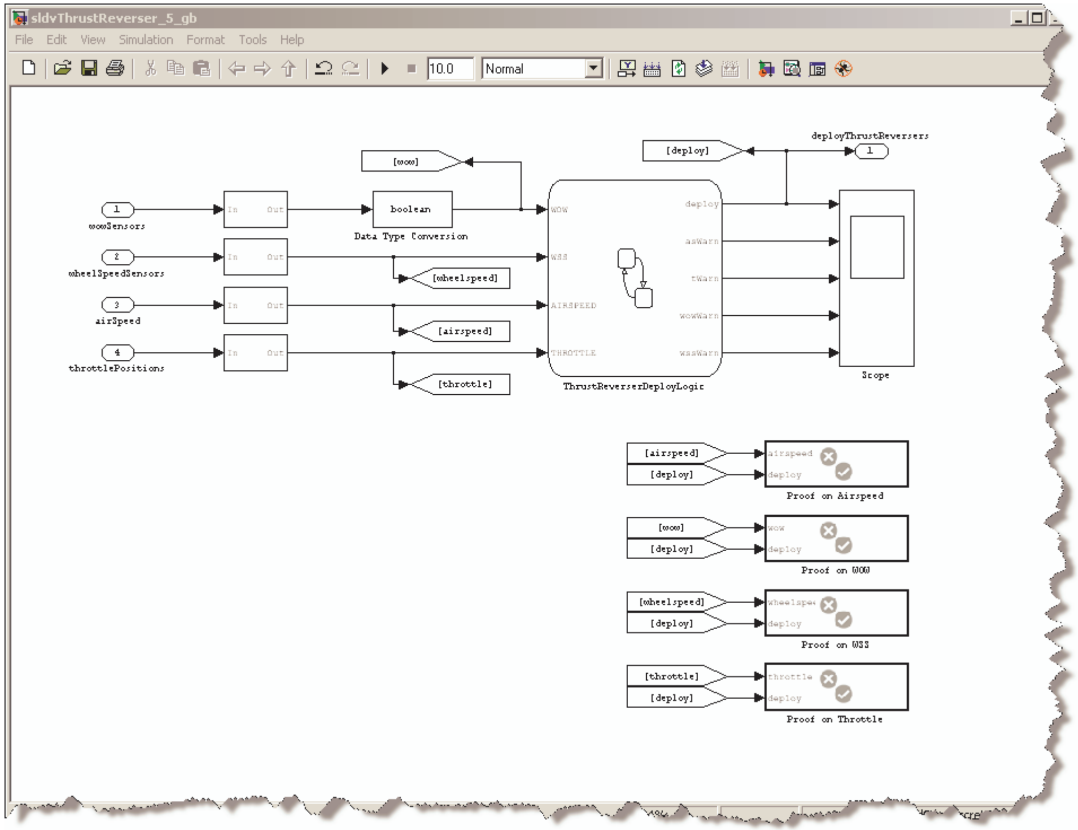
- Design (Structure) ->



- Design (Behavior) ->

Demonstration

- Demonstration of Property Proving with Simulink Design Verifier

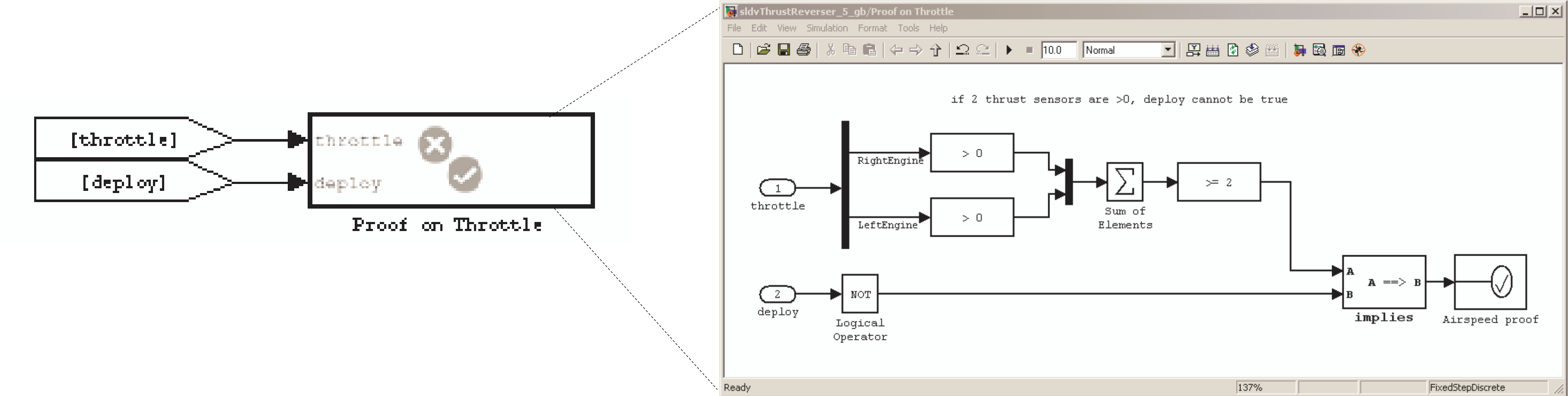


Modeling Functional Requirements

Simulink Design Verifier

Functional Requirement:

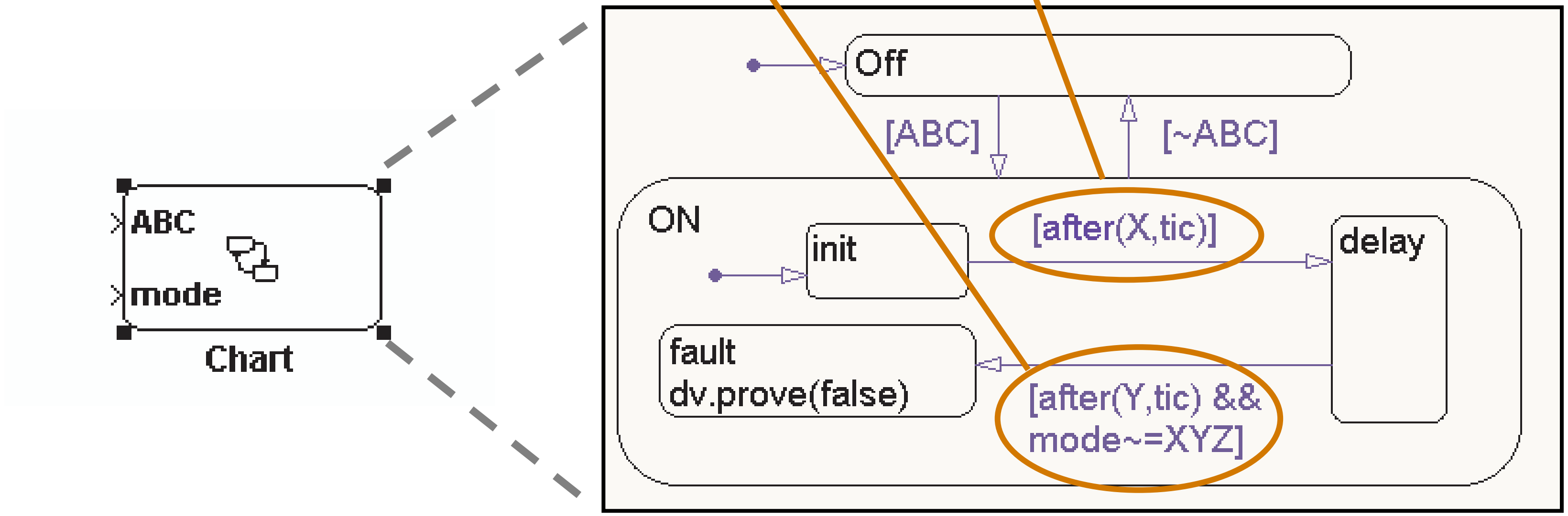
- If 2 or more thrust sensors are >0, the thrust reverser will not deploy



Modeling Functional Requirements

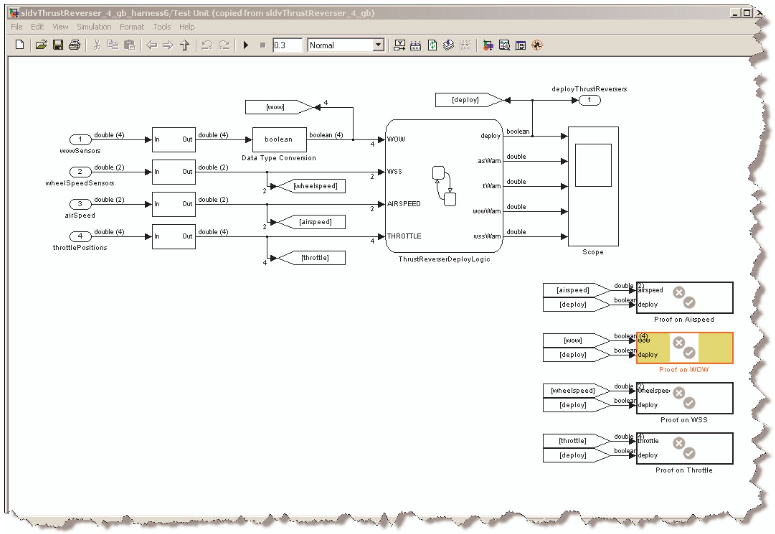
Expressing requirements with temporal aspects

After condition ABC is true ***for X sample periods*** the controller shall enter mode XYZ ***within Y samples***.

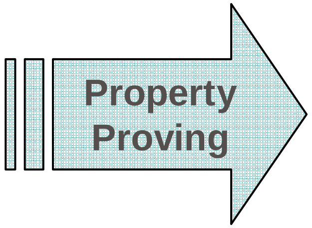


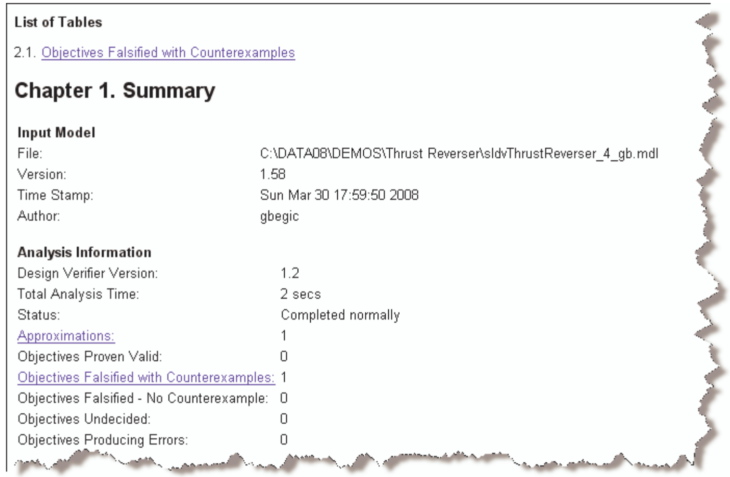
Proving Design Properties

Simulink Design Verifier



Property Proving Harness augmented with design properties





List of Tables

2.1. Objectives Falsified with Counterexamples

Chapter 1. Summary

Input Model

File:	C:\DATA08\DEMOS\Thrust Reverser\sldvThrustReverser_4_gb.mdl
Version:	1.58
Time Stamp:	Sun Mar 30 17:59:50 2008
Author:	gbegic

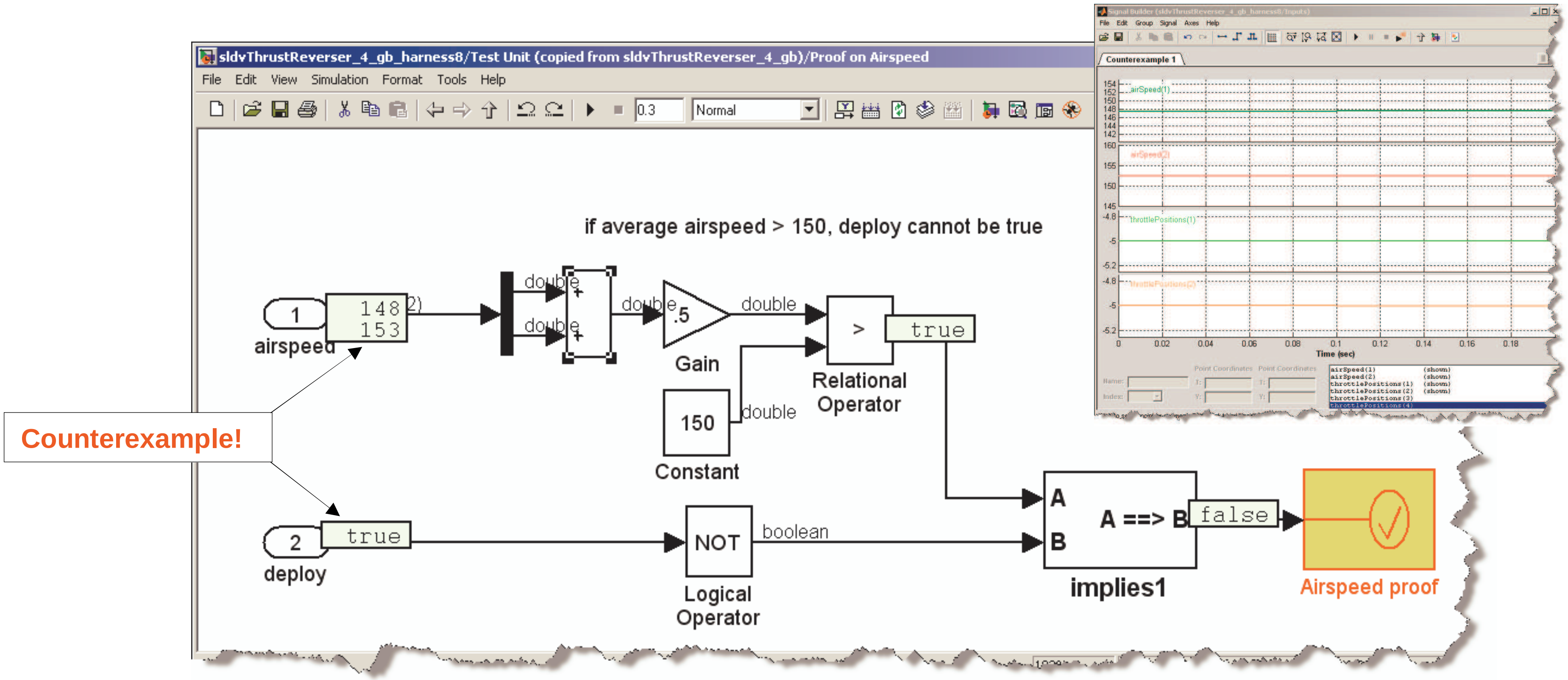
Analysis Information

Design Verifier Version:	1.2
Total Analysis Time:	2 secs
Status:	Completed normally
Approximations:	1
Objectives Proven Valid:	0
Objectives Falsified with Counterexamples:	1
Objectives Falsified - No Counterexample:	0
Objectives Undecided:	0
Objectives Producing Errors:	0

Detailed report and violations

Property Proving - Counterexample

- Leads to improvement of design and/or requirements

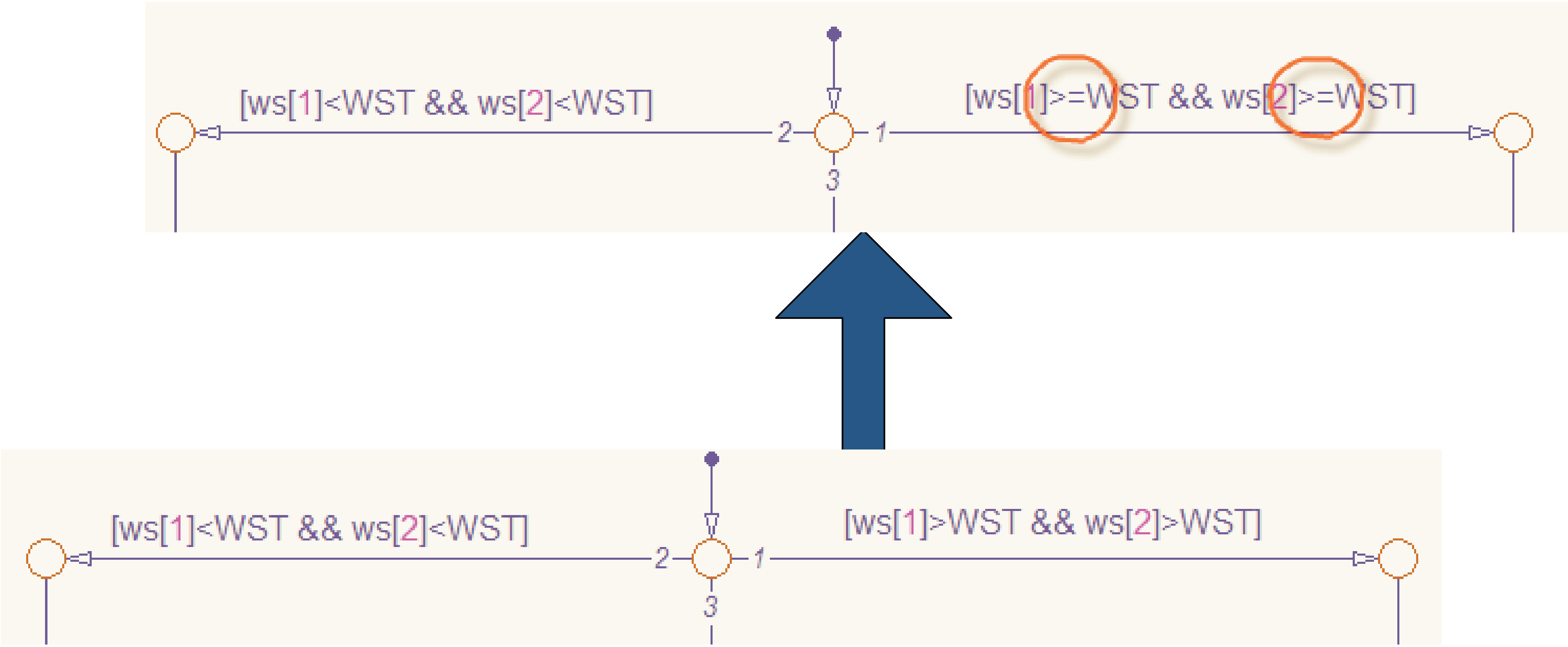


Improvements

- After some quality design time…

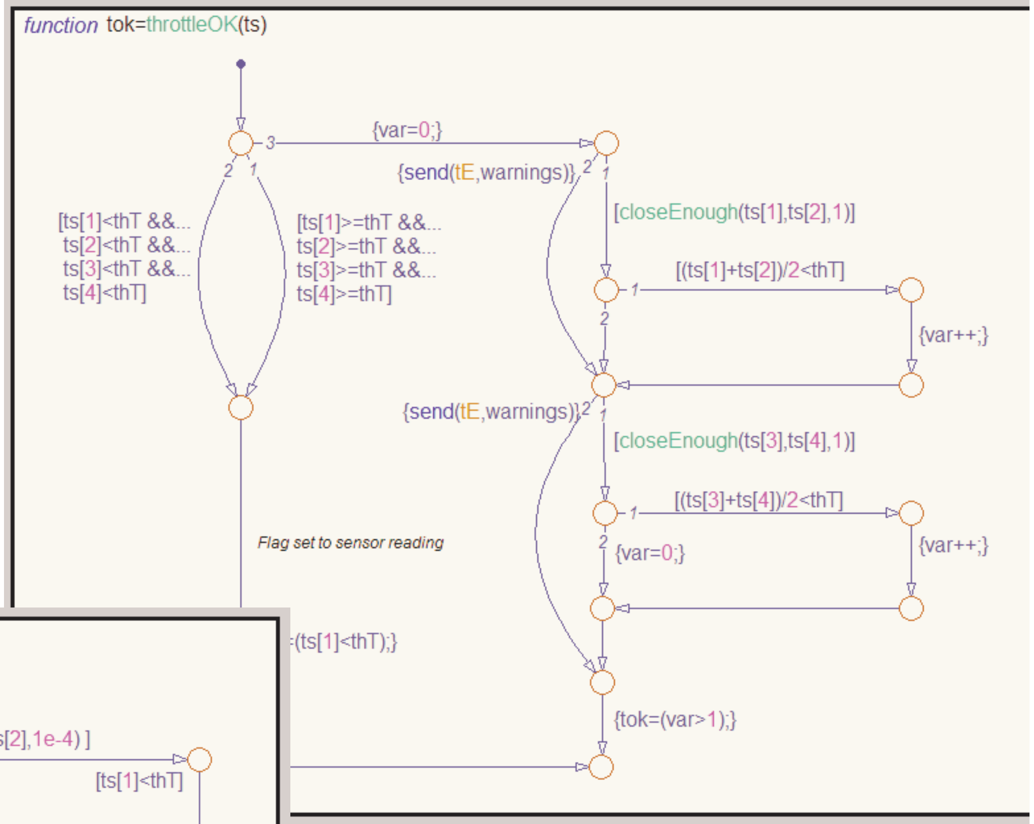
Wheel Speed Check Errors

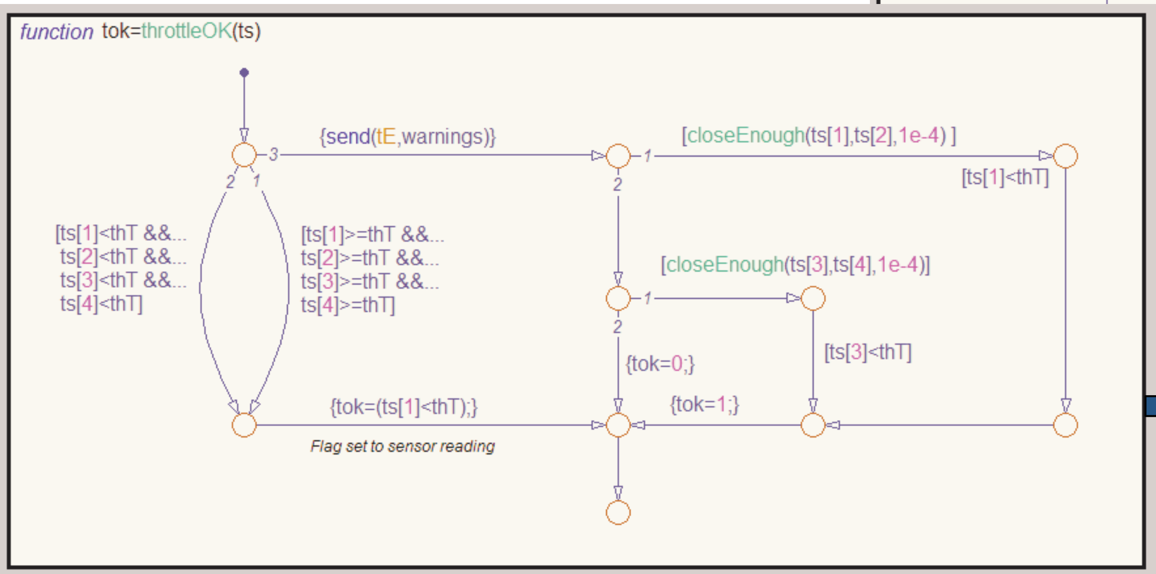
- Forgot the “=“ case



Throttle Logic Significantly Flawed

- What if 1 throttle is higher than the threshold, and 1 is lower?





Proving Properties – Workflows

Simulink Design Verifier

1. Authoring
   - Highly Iterative
   - Leads to improvement in design and in specifications
2. Execution and Reporting
   - Automated
   - Part of the regression testing harness

- **Benefits**
  - Leads to precise definition of low level functional requirements
  - Once established properties represent a model of design behavior
  - Minimizes a chance of implementing undesired behavior

Closing Remarks

Verification and Validation Tools

Requirements

Design
- Environment
- Physical Components
- Algorithms

Hand-Generate
Generate
Generate

Implement
Digital Electronics: VHDL, Verilog

Embedded Software: C, C++

FPGA | ASIC | MCU | DSP

Integration

System V and V
- Robustness Testing
- Modeling Standards Checking
- Requirements Validation

SystemTest
Simulink Verification and Validation
xPC Target

Component V and V
Design Verification
- Model Testing
- Coverage and Test Generation
- Property Proving

SystemTest
Simulink Verification and Validation
Simulink Design Verifier

Code Verification
- Code Correctness
- Processor-In-The Loop Testing

PolySpace products
Embedded IDE Link products
Target Support Package products

Integration Testing
- Software Integration Testing
- Hardware-in-the-Loop Testing
- Hardware Connectivity

xPC Target
Data Acquisition Toolbox
Instrument Control Toolbox

Do I Need To Implement All / Some of the New Verification and Validation Methods?

- Traditional Verification and Validation Methods
  - Hardware Integration Testing
  - Software Integration Testing
  - Unit Testing of Code
  - Ad-hoc Testing in Simulation

- Methods for Early Verification and Validation
  - Traceability
  - Modeling and Coding Standards Checking
  - Model Testing
  - Proving Design Properties and Code Correctness

Motorola Creates Electric Vehicle Battery Management Controller with Real-Time Workshop Embedded Coder


The Motorola electronic control unit

Challenge

To develop battery management controller software within a tight deadline

Solution

Use integrated tools for Model-Based Design and code generation from The MathWorks to design, test, and manage requirements for the controller

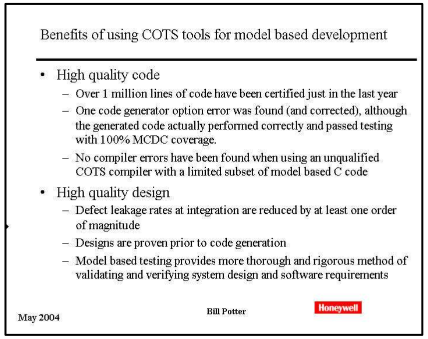
Results

- Automatic generation of efficient C code
- Optimized memory resources
- Ability to detect design flaws before generating code

To validate the design against the customer's requirements, the engineers associated the model components to the written requirements with the Requirements Management Interface. "Internal reviews were then easy, and we could demonstrate to our customer that all the requirements had been met."

Salam Zeidan
Software Manager
Motorola Automotive

Model-Based Design for Safety-Critical Applications Success Stories



Benefits of using COTS tools for model based development

- High quality code
  - Over 1 million lines of code have been certified just in the last year
  - One code generator option error was found (and corrected), although the generated code actually performed correctly and passed testing with 100% MCDC coverage.
  - No compiler errors have been found when using an unqualified COTS compiler with a limited subset of model based C code
- High quality design
  - Defect leakage rates at integration are reduced by at least one order of magnitude
  - Designs are proven prior to code generation
  - Model based testing provides more thorough and rigorous method of validating and verifying system design and software requirements

May 2004 Bill Potter Honeywell

Honeywell Generates DO-178B Certified Code

- 1,000,000+ lines of code certified in a single year
- 6.3 sigma quality achieved

Alstom Generates Production Code for Safety-Critical Power Converter Control Systems

- Defect-free, safety-critical code generated and certified
- Development time cut by 50 percent

“the railway application was the first with automatically generated code to receive TÜV certification.”

Institute for Radiological Protection and Nuclear Safety Verifies Nuclear Safety Software with PolySpace™ Products for C/C++

Summary

- Model-Based Design is a platform that enables you to start verification and validation of designs and embedded software early
- When building a verification environment for your models and the generated code there are several different methods you can use to increase confidence in your designs
  - Traceability
  - Modeling and Coding Standards checking
  - Testing
  - Proving
- The MathWorks consulting and training teams can help you create a plan for the optimization of your verification and validation process