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

Requirements

Design
- Environment
- Physical Components
- Algorithms
- Digital Electronics
- Embedded Software
  - VHDL, Verilog
  - C, C++
  - FPGA, ASIC, MCU, DSP

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

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

Integration Testing
- Code Verification
  - Code Correctness
  - Processor-In-The-Loop Testing
- Software Integration Testing
  - Hardware-in-the-Loop Testing
Address the Entire Development Process

Requirements

Design
- Environment
- Physical Components
- Algorithms

Digital Electronics
- VHDL, Verilog

Embedded Software
- C, C++
- FPGA
- ASIC
- MCU
- DSP

Implement

Integration

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

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

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

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

Master Class
 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
- Environment
- Physical Components
- Algorithms

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

Adopting Model-Based Design within Aerospace and Defense
System V and V - Example

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

Functional Requirements

Design
- Environment
- Physical Components
- Algorithms

Design Verification
- Analysis Model
- C
- Code Harness
- Component Source Code
- Code Generation
- Detailed models

Implementation
- FPGA
- ASIC
- MCU
- DSP
- VHDL, Verilog
- C, C++

Hand-Generate, Generate, Generate

Algorithm
- Functional Requirements
- Generate
- Generate
- Generate

Test Application
- Detailed models
- Code Harress
Model Coverage
Simulink Verification and Validation

- Structural metric
- Measure of test completeness

Example MC/DC Coverage

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

Example MC/DC Coverage

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

<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% (6/4) condition outcomes</td>
</tr>
<tr>
<td>MCDC (C1)</td>
<td>50% (1/2) conditions reversed the outcome</td>
</tr>
</tbody>
</table>

Conditions analyzed:

<table>
<thead>
<tr>
<th>Description</th>
<th>True</th>
<th>False</th>
</tr>
</thead>
<tbody>
<tr>
<td>input part 1</td>
<td>8</td>
<td>3</td>
</tr>
<tr>
<td>input part 2</td>
<td>3</td>
<td>6</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Decision/Condition</th>
<th>True Out</th>
<th>False Out</th>
</tr>
</thead>
<tbody>
<tr>
<td>expression for output</td>
<td></td>
<td></td>
</tr>
<tr>
<td>input part 1</td>
<td>TT</td>
<td>(FT)</td>
</tr>
<tr>
<td>input part 2</td>
<td>TT</td>
<td>TF</td>
</tr>
</tbody>
</table>
Objectives for Test Generation
Simulink Design Verifier

Chapter 2. Test Objectives

Table of Contents

Status
coverage example

Table 2.1. Objectives Satisfied

<table>
<thead>
<tr>
<th>#</th>
<th>Type</th>
<th>Model Item</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Decision</td>
<td>Switch</td>
<td>Switch “Switch”; logical trigger input false (output is from 3rd input port)</td>
</tr>
<tr>
<td>2</td>
<td>Decision</td>
<td>Switch</td>
<td>Switch “Switch”; logical trigger input true (output is from 1st input port)</td>
</tr>
<tr>
<td>3</td>
<td>Condition</td>
<td>Logical Operator</td>
<td>Logic “Logical Operator”; input port 1 T</td>
</tr>
<tr>
<td>4</td>
<td>Condition</td>
<td>Logical Operator</td>
<td>Logic “Logical Operator”; input port 1 F</td>
</tr>
<tr>
<td>5</td>
<td>Condition</td>
<td>Logical Operator</td>
<td>Logic “Logical Operator”; input port 2 T</td>
</tr>
<tr>
<td>6</td>
<td>Condition</td>
<td>Logical Operator</td>
<td>Logic “Logical Operator”; input port 2 F</td>
</tr>
<tr>
<td>7</td>
<td>Mcdc</td>
<td>Logical Operator</td>
<td>Logic “Logical Operator”, MCDC expression for output with input port 1 T</td>
</tr>
<tr>
<td>8</td>
<td>Mcdc</td>
<td>Logical Operator</td>
<td>Logic “Logical Operator”, MCDC expression for output with input port 1 F</td>
</tr>
<tr>
<td>9</td>
<td>Mcdc</td>
<td>Logical Operator</td>
<td>Logic “Logical Operator”, MCDC expression for output with input port 2 T</td>
</tr>
<tr>
<td>10</td>
<td>Mcdc</td>
<td>Logical Operator</td>
<td>Logic “Logical Operator”, MCDC expression for output with input port 2 F</td>
</tr>
</tbody>
</table>

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

Affects (X & Y) to be T and 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.

Test Cases are captured in a Signal Builder block.

Input data sequences drive system from its initial configuration.

Original model copied to the harness.
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®)

- 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

U.S. Orders Thrust Reverser Deactivated on 767s

By Barry James

PARIS: The Federal Aviation Administration in Washington ordered U "deactivate" engine thrust reversers on Boeing 767 jetliners. Such a device may have been one of the causes of the 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.

Lauda Air B767 Accident Report

SYNOPSIS

Prepared for the WWW by

Hiroshi Sogame
Safety Promotion Comt.
All Nippon Airways
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

- Functional Requirements
- Augmented Component Model
- Model Harness
- Analysis Report

Design
- Hand-Generate
- Generate
- Generate
- VHDL, Verilog
- C, C++
- FPGA
- ASIC
- MCU
- DSP

Integration

Implement
Property Proving – Overview
Simulink Design Verifier

- Design (Structure) ->

- Design (Behavior) ->
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

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

Implement

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

PolySpace products
Embedded IDE Link products
Target Support Package products

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

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

xPC Target
Data Acquisition Toolbox
Instrument Control Toolbox

Digital Electronics
- VHDL, Verilog
- FPGA

Embedded Software
- C, C++
- ASIC
- MCU
- DSP
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

**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

The Motorola electronic control unit
Model-Based Design for Safety-Critical Applications

Success Stories

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.