Verification and Validation of Models and Code

Presenter: Mark Walker
mark.walker@mathworks.co.uk
Agenda

- Introductions
- Workflows for verification and validation
Introductions

- I spend most of my time:

  A. Creating specifications and requirements (systems and software)

  B. Implementation based on specification and requirements created by somebody else (generating / writing / deploying / debugging code)

  C. Other (including both, or none of the above)
How much time do we need to get 100% MC/DC coverage?
Costs of Embedded Software Fault Propagation

<table>
<thead>
<tr>
<th>Defect Type</th>
<th>Relative Cost to Fix Defects per Phase Found</th>
</tr>
</thead>
<tbody>
<tr>
<td>Requirements</td>
<td>50</td>
</tr>
<tr>
<td>Design</td>
<td>45</td>
</tr>
<tr>
<td>Code</td>
<td>40</td>
</tr>
<tr>
<td>Test</td>
<td>35</td>
</tr>
</tbody>
</table>


Cost of fixing defects detected depending on where they are introduced.
Methods for Early Verification and Validation

- Traceability
  - Requirements to model and code
  - Model to code

- Modeling and Coding Standards
  - Modeling standards checking
  - Coding standards checking

- Testing
  - Model testing in simulation
  - Processor In the loop

- Proving
  - Proving design properties
  - Proving code correctness
Increasing Confidence In Your Designs

![Chart showing the increase in confidence with verification methods]

- Traceability
- Modeling and Coding Standards Checking
- Model and Code Testing
- Proving Design Properties and Code Correctness

Verification Method
Address the Entire Development Process

**Design**
- Environment
- Physical Components
- Algorithms
- VHDL, Verilog
- C, C++
- FPGA, ASIC, MCU, DSP

**Integration**
- Generate
- Implement

**System V&V**
- Requirements Validation
- Robustness Testing
- Modeling Standards Checking

**Component V&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
- Hardware Connectivity
Traceability

- **Tracing Requirements ↔ Model**
  Simulink® Verification and Validation™

- **Tracing Model ↔ Source Code**
  Real-Time Workshop® Embedded Coder™

- **Tracing Requirements ↔ Source Code**
  Simulink Verification and Validation
Modeling and Coding Standards

- Coding Standards Checking
  Simulink Verification and Validation

- Coding Standards Checking
  PolySpace™ Client™ for C/C++
Early Validation and Robustness Testing
Component Testing

**Functional Requirements**

- Design
- Design Verification
- Code Verification

**Environment**

- Physical Components
- Algorithms

**Digital Electronics**

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

**Embedded Software**

- Generate

**Integration**

- Implement

MathWorks
Aerospace and Defence Conference '08
Test Generation Workflow

Functional Requirements

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

Digital Electronics
- Environment
- Physical Components
- Algorithms

Embedded Software

Integration

Implement

Code Generation
- Detailed models

Code Harness

Analysis Model
- C

Test Application
- Component Source Code

MathWorks Aerospace and Defence Conference '08
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
Demo

- Processor-in-the-loop co-simulation
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
Code Correctness

Formal method:
Abstract Interpretation

Results are proven for all possible executions of the code!!
Code Correctness

- A model is a well controlled way to specify system behaviour
  - Generated code matches the model
  - Few ambiguities, low warning rate
  - 100% green is a realistic target
Demo

- Proving a functional requirement
Example Problems vs. Tools

- Incorrect Dynamic Response
  - Simulation Testing
  - Rapid Prototyping and Hardware-in-the-Loop

- Model Error: \( \max(a,b) \) instead of \( \min(a,b) \) to apply upper clip
  - Simulation Testing
  - Property proving with Simulink Design Verifier
Example Problems vs. Tools

- Unreachable state / transition / code
  - Test generation with Simulink Design Verifier
  - PolySpace

- Overflow / underflow
  - Simulation
  - PolySpace

- Execution time exceeds deadline
  - Simulation (requires execution time model)
  - Processor-in-the-loop

MathWorks
Aerospace and Defence Conference '08
Summary

- Model-Based Design enables early verification and validation!

- Early verification and validation methods improve and optimize your existing development process.

- Early problem detection significantly reduces time spent debugging – shorter time to resolution
Master Class Invitation

- Methods for Early Verification and Validation
  - Robustness Testing
  - Automatic Test Generation
  - Property Proving