Verification and Validation of Models and Code

Presenter: Goran Begic
Technical Marketing
Goran.Begic@mathworks.com
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)
Great Demo

- How much time do we need to get 100% MC/DC coverage?
Address the Entire Development Process

Requirements

**Design**
- Environment
- Physical Components
- Algorithms

**Digital Electronics**
- VHDL, Verilog
- FPGA

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

**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
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

Verification Method

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

Confidence
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

MathWorks Symposium
Adopting Model-Based Design within Aerospace and Defense
Modeling and Coding Standards

- Modeling Standards Checking
  Simulink Verification and Validation

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

Requirements

Design
- Environment
- Physical Components
- Algorithms

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

MATLAB & SIMULINK

MathWorks Symposium
Adopting Model-Based Design
within Aerospace and Defense
Component Testing

Functional Requirements

Design
- Environment
- Physical Components
- Algorithms

Digital Electronics
- VHDL, Verilog
- FPGA
- ASIC

Embedded Software
- C, C++

Integrated Software
- MCU
- DSP

Hand-Generate

Design Verification

Code Verification

MathWorks Symposium
Adopting Model-Based Design within Aerospace and Defense
Test Generation Workflow

Functional Requirements

Design

- Design Verification
- Code Verification

Digital Electronics
- VHDL, Verilog
- FPGA
- ASIC

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

Physical Components

Algorithms

Hand-Generate

Generate

Generate

Analysis Model

Detailed models

Code Generation

Code Harness

Component Source Code

Test Application

Implement

Integration

MathWorks Symposium
Adopting Model-Based Design within Aerospace and Defense
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
**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

```
static void Pointer_Arithmetic ()
{
    int tab[100];
    int i, *p = tab;

    for(i = 0; i < 100; i++, p++)
        *p = 0;

    if(get_bus_status() > 0)
    {
        if(get_oil_pressure() > 0)
        {
            *p = 5; /* Out of bounds */
        }
        else
        {
            i++;
        }
    }

    i = random_int();
    if (random_int()) *(p-i) = 10;

    if (0<i && i<=100)
    {
        p = p - i;
        *p = 5; /* Safe pointer access */
    }
}
```
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