MATLAB EXPO 2016
Increasing Design Confidence

Paolo Bizzarri
Principal Applications Engineer
The Cost of Failure…

Ariane 5
$7,500,000,000
Rocket & payload lost

USS Yorktown
0 Knots
Top speed

Therac-25
6 Casualties
due to radiation overdose
Key Message

It is easier and less expensive to fix design errors early in the process when they happen.

Model and code verification enable:

1. Early testing to increase confidence in your design
2. Delivery of higher quality software throughout the workflow
3. Greater safety!
Gaining Confidence in our Design

- Ad-hoc testing
- Design error detection
- Functional & structural tests
- Modeling standards
- Model & code equivalence checks
- Code integration analysis

Confidence vs. Effort / Time
Application: Control of electromechanical systems
Application: Motor Control

ECU / Control Unit

1. Overall Control Unit
2. Motor Control (MBD)
3. Diagnostics
4. Filtering

System Inputs

Legacy code

Outputs
Application: Electromechanical Control System
Application: Motor Control

**Inputs**

**Systems Inputs**
- Motor On
- Command Type
- Command Value

**Sensor Inputs**
- Currents, Voltages
- Encoders

**Outputs**
- Engaged
- Target speed

Motor Control (MBD)

Position
Speed
Torque
Gaining Confidence in our Design

- Ad-hoc testing
- Design error detection
- Functional & structural tests
- Modeling & coding standards
- Code equivalence & integration checks
Ad-hoc Tests

New “Dashboard” blocks facilitate early ad-hoc testing
Gaining Confidence in our Design

Confidence

Effort / Time

Ad-hoc testing

Modeling & coding standards

Code equiv. & integration checks

Design error detection

Functional & structural tests

Gaining Confidence in our Design

Confidence

Effort / Time

Ad-hoc testing

Modeling & coding standards

Code equiv. & integration checks

Design error detection

Functional & structural tests
# Finding Design Errors: Dead Logic

## TABLE C-1.3: PARAMETERS

<table>
<thead>
<tr>
<th>NAME</th>
<th>DIM</th>
<th>TYPE</th>
<th>FREQ.</th>
<th>UNITS</th>
</tr>
</thead>
<tbody>
<tr>
<td>CtrlParams with FIELDS</td>
<td>1</td>
<td>STRUCT</td>
<td>200Hz</td>
<td>Various, see details</td>
</tr>
<tr>
<td>Current P: 10</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Current I: 10000</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Velocity P: 0.0050</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Velocity I: 0.0150</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Position P: 0.1000</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Position I: 0.6000</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Startup Acceleration: 1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Startup Current: 0.2000</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Ramp To Stop Velocity: 0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>AdcZeroOffsetDriverUnits: 2.2523e+03</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>AdcDriverUnitsToAmps: 0.0049</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>EncoderToMechanicalZeroOffsetRads: 0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PmsmPolePairs: 4</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Gaining Confidence in our Design

- Ad-hoc testing
- Design error detection
- Effort / Time

Confidence
Simulation Testing Workflow

Requirements

Design

Did we meet requirements?

Review functional behavior

Functional Structural

MATLAB EXPO 2016
Did We Completely Test our Model?

Potential causes of less than 100% coverage:

- Missing requirements
- Over/Under-specified design
- Design errors
- Missing tests
Requirements Based Functional Testing with Coverage Analysis

Contents
- Clear all the test space
- NUMBER OF TEST
- SIGNAL BUILDER BLOCKS UNDER TEST
- CONTROLLED OUTPUTS
- TEST NAMES
- CYCLE ON TEST AND COMPARISON

Clear all the test space

NUMBER OF TEST
NTEST=6;

SIGNAL BUILDER BLOCKS UNDER TEST

CONTROLLED OUTPUTS
TEST NAMES

test[1]="0LoadVelocityControl1";
test[2]="VelocitySpecOff";
test[3]="NormTest2";

TEST # 2: VelocityOnOff

TEST # 3: TorqueStep

MATLAB EXPO 2016
Functional Testing with Added Requirements & Test Cases
Gaining Confidence in our Design

- Found a major design problems thanks to **Simulink Design Verifier**
  - A positive number was always evaluated as less than zero
- Functional test cases passed
  - **Simulink Test** or Classic Signal Builder manage execution of tests
- Generated extra test cases for better understanding how to pilot the control logic
  - During test cases generation another defect was corrected (% commented transition)
- With some effort **100% model coverage was reached!**
Model Advisor – Model Standards Checking

Modeling Standards for IEC 61508

- Check model object names
  - Identify invalid names of following model objects (first invalid name fragment is highlighted):
    - Blocks
    - Signals
    - Parameters

Check for root inputs with missing range definitions

- Identify root-level Import blocks with missing or erroneous minimum or maximum values. Import block minimum
- with block parameters or Simulink signal objects that explicitly resolve to the connected signal lines.

Warning
- This check is only supported at the model level.

Recommended Action
- To run this analysis, please open the model advisor from the top level of the model instead of the subsystem level.

See Also
- IEC 61508-3, Table B.9 (6) – Fully defined interface
- IEC 62304, 5.5.3 - Software Unit acceptance criteria
- ISO 26262-6, Table 2 (2) – Precisely defined interfaces
- EN 50128, Table A.1.11 – Software Interface Specifications, Table A.3.19 – Fully Defined Interface
- hist_0025: Design min/max specification of input interfaces
Gaining Confidence in our Design

- Ad-hoc testing
- Design error detection
- Functional & structural tests
- Modeling standards

Confidence vs. Effort / Time
Code Generation with Model-to-Code Traceability

Mode Scheduler

Configures controller mode and controller inputs based on user commands.
Code Generation with Model-to-Code Traceability

SPECS  MODEL  CODE

CONTROL AND MANAGE COMPLEXITY
Equivalence Testing:
Model vs SIL or PIL Mode Testing

Coverage $\rightarrow$ 100%

Model Testing

Test vectors $i(t)$

SIL or PIL Mode Testing

Object code

Execution

Result vectors $o_{code}(t)$

Target compiler and linker

Generated C code

Embedded Coder

Model used for production code generation

Simulation

Result vectors (base line) $o_{sim}(t)$
Code Equivalence Check Results: Model vs Code

- Re-used full coverage test vectors and harnesses from Model Verification testing
- Ran test vectors on generated code using Model Reference SIL mode
- Model Coverage to Code Coverage using the SIL Code Coverage Report
- Successfully demonstrated code behavior matches model behavior!
Gaining Confidence in our Design

- Ad-hoc testing
- Design error detection
- Functional & structural tests
- Modeling standards
- Model & code equivalence checks
- Code integration analysis

Confidence vs. Effort / Time
Source code painted in green, red, gray, orange

Green: reliable
Safe pointer access

Red: faulty
Out of bounds error

Gray: dead
Unreachable code

Orange: unproven
May be unsafe for some conditions

Purple: violation
MISRA-C/C++ or JSF++ code rules

Range data
Tool tip

CODE VERIFICATION TOOLS: Bug Finder and Code Prover
Polyspace Code Analysis is STATIC CODE ANALYSIS

static void pointer_arithmetic (void) {
    int array[100];
    int *p = array;
    int i;
    for (i = 0; i < 100; i++) {
        *p = 0;
        p ++;
    }
    if (get_bus_status () > 0) {
        if (get_oil_pressure () > 0) {
            *p = 5;
        } else {
            i ++;
        }
    }
    i = get_bus_status();
    if (i >= 0) {
        *p - i = 10;
    }
}
RESULTS OF BUG FINDER: errors and MISRA rules

<table>
<thead>
<tr>
<th>Name</th>
<th>BlockType</th>
</tr>
</thead>
<tbody>
<tr>
<td>Solver</td>
<td></td>
</tr>
<tr>
<td>Simulation Target</td>
<td></td>
</tr>
<tr>
<td>Simscape</td>
<td></td>
</tr>
<tr>
<td>Polyspace</td>
<td></td>
</tr>
<tr>
<td>Optimization</td>
<td></td>
</tr>
<tr>
<td>Model Referencing</td>
<td></td>
</tr>
<tr>
<td>HDL Code Generation</td>
<td></td>
</tr>
<tr>
<td>Hardware Implement</td>
<td></td>
</tr>
<tr>
<td>Diagnostics</td>
<td></td>
</tr>
<tr>
<td>Design Verifier</td>
<td></td>
</tr>
<tr>
<td>Data Import/Export</td>
<td></td>
</tr>
<tr>
<td>Code Generation</td>
<td></td>
</tr>
</tbody>
</table>

**Code Generation**

- **Code Style**
  - Parentheses level: Maximum (Specify precedence with parentheses)
  - Preserve operand order in expression
  - Preserve condition expression in if statement
  - Convert if-elseif-else patterns to switch-case statements
  - Preserve extern keyword in function declarations
  - Suppress generation of default cases for Stateflow switch statements if unreachable
  - Replace multiplications by powers of two with signed bitwise shifts
  - Allow right shifts on signed integers
  - Casting modes: Nominal
  - Indent style: ```Indent size: 2```

**MISRA 2012**
RESULTS OF CODE PROVER: **absolute absence of run time errors**
Gaining Confidence in our Design

**Bug Finder** finds only two MISRA violations
- These violations can be addressed with another transition
  - OR could be addressed with a different style of code generation
- In both cases the re-validation is automatized

**Code Prover** initially finds possible overflows
- If we run analysis with correct input ranges the code is “all green” → no RTE
Polyspace Products proven on SOLAR IMPULSE

Avionics Verified and Validated with Polyspace

Autopilot Verified and Validated with Model-Based Design

Visit me at the code verification demo booth .....
Conclusion

It is easier and less expensive to fix design errors early in the process when they happen.

Model and code verification enable:

1. Early testing to increase confidence in your design
2. Delivery of higher quality software throughout the workflow
3. Greater safety!