MATLAB EXPO 2016
Increasing Design Confidence
(Aumentare l’affidabilita’ dei progetti)

Paolo Bizzarri
Principal Application 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

MATLAB EXPO 2016
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

MODEL  C/C++ CODE

Confidence  Effort / Time

MATLAB EXPO 2016
Application: Control of electromechanical systems
Application: Motor Control

ECU / Control Unit

- Overall Control Unit
- Motor Control (MBD)
- Diagnostics
- Filtering

System Inputs

Legacy code

Outputs
Application: Electromechanical Control System

Overall Control Unit

Motor Control (MBD)

Diagnostics

Filters

System Inputs

Legacy code

Outputs
Application: Motor Control

Inputs

Systems Inputs
- Motor On
- Command Type
- Command Value

Sensor Inputs
- Currents, Voltages
- Encoders

Motor Control (MBD)

Outputs
- Engaged
- Target speed
Gaining Confidence in our Design

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

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

Confidence

Effort / Time

Ad-hoc testing
### 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>StartupAcceleration: 1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>StartupCurrent: 0.2000</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RampToStopVelocity: 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
- Functional & structural tests
- Modeling & coding standards
- Code equivalence & integration checks
Simulation Testing Workflow

**Requirements**

Did we meet requirements?

**Design**

**Review functional behavior**

**Functional Structural**

**MATLAB EXPO 2016**
Did We Completely Test our Model?

Model Coverage Analysis

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

```
% Vito Bizzarri 2013
clear all;close all;close all;
modelname='TestHarnessGlobal';
load_system(modelname);
```

NUMBER OF TEST

```
NTEST=6;
```

SIGNAL BUILDER BLOCKS UNDER TEST

```
blk_input='TestHarnessGlobal/PerformanceTests';
blk_expected='TestHarnessGlobal/Expected';
```

CONTROLLED OUTPUTS

TEST NAMES

```
test(1,1)='VelocityVelocityOnOff';
test(2,1)='velocityOnOff';
test(3,1)='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

Model Advisor Report - rtwdemo_psmfoc.slx

Simulink version: 8.7
System: rtwdemo_psmfoc/Mode_Scheduler/Controller_Mode_Scheduler
Treat as Referenced Model: off

Run Summary

<table>
<thead>
<tr>
<th></th>
<th>Pass</th>
<th>Fail</th>
<th>Warning</th>
<th>Not Run</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>14</td>
<td>0</td>
<td>14</td>
<td>0</td>
<td>28</td>
</tr>
</tbody>
</table>

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 Inports with missing range definitions

Identify root-level Inport blocks with missing or erroneous minimum or maximum values. Inport block minimum and maximum values should be specified, 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
hisl_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
Code Generation with Model-to-Code Traceability

SPECS  MODEL  CODE

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

Coverage → 100%
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
CODE VERIFICATION TOOLS: Bug Finder and Code Prover Polyspace Code Analysis is STATIC CODE ANALYSIS

Source code painted in green, red, gray, orange

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++;
        }
    } else {
        i = get_bus_status();
        if (i >= 0) {
            *p-i = 10;
        }
    }
}
RESULTS OF BUG FINDER: errors and MISRA rules

MISRA 2012
RESULTS OF CODE PROVER: absolute absence of run time errors

NO RTE (Run Time Error Code)
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
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!