Increasing Design Confidence

Model and Code Verification
The Cost of Failure...

Ariane 5

$7,500,000,000

Rocket & payload lost
The Cost of Failure…

USS Yorktown

0 Knots

Top speed
The Cost of Failure…

Therac-25

6 Casualties

due to radiation overdose
Motivation

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

Model-Based Design enables:

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

- Ad-hoc testing
- Design error detection
- Functional & structural tests
- Modeling standards
- Model & code equivalence checks
- Code integration analysis
Application: Cruise Control
Control speed according to setpoint
Application: Cruise Control

ECU

1. Cruise Control Module (MBD)
2. Fuel Rate Control Module
3. Shift Logic Control Module

Legacy code

System Inputs

ECU system

Outputs
Application: Cruise Control

ECU

- Cruise Control Module (MBD)
- Fuel Rate Control Module
- Shift Logic Control Module

System Inputs

Legacy code

Outputs
Application: Cruise Control

Inputs
- Cruise_onoff
- Brake
- Speed
- Coast set
- Accel reset

Cruise Control Module (MBD)

Outputs
- Engaged
- Target speed
Gaining Confidence in our Design

- Ad-hoc testing
- Modeling & coding standards
- Code equivalent & integration checks

Confidence vs. Effort / Time
Ad-hoc Tests

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

Confidence

Effort / Time

Ad-hoc testing

Design error detection

Functional & structural tests

Modeling & coding standards

Code equiv. & integration checks
Finding Design Errors: Dead Logic

Design Error Detection:
- Dead logic
- Identify active logic

Results:
- Design error detection completed normally.
- 2/70 objectives are dead logic.
- 68/70 objectives are active logic.

- Generate detailed analysis report
- Open harness model

Transition: Transition trigger expression F DEAD LOGIC
Transition: Transition trigger expression T ACTIVE LOGIC

[after(incdec/holdrate... *10, tick)]
Finding Unintended Behavior

- Dead logic due to “uint8” operation on \( \text{incdec}/\text{holdrate} \times 10 \)
- Fix change the order of operation \( 10 \times \text{incdec}/\text{holdrate} \)
Gaining Confidence in our Design

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

**Requirements**

**Design**

**Did we meet requirements?**

**Review functional behavior**

**Did we completely test our model?**

**Structural coverage report**
Did We Completely Test our Model?

Model Coverage Analysis

Potential causes of less than 100% coverage:
- Missing requirements
- Over-specified design
- Design errors
- Missing tests
Requirements Based Functional Testing with Coverage Analysis

- All 14 requirements based test cases pass
- By analyzing model coverage results we find:
  - Missing test cases for vehicle speed exit conditions, and
  - Missing requirements (and test cases) for “hold” or continuous speed button input
Functional Testing with Added Requirements & Test Cases
Functional Testing with Added Requirements & Test Cases

- Added 2 new requirements for the “hold” case for speed setting input buttons
- Added 5 test cases to the original 14 requirements based test cases
  - 3 test cases for the 2 new requirements
  - 2 test cases for the missing test cases for the vehicle speed exist conditions
- 4/5 new functional test cases pass
  - Failed test case showed overshoot beyond target speed limits
  - Coverage analysis highlighted transitions with design errors
  - Fixed comparison operators, (<) → (<=), and (>) → (>=)
- Now all (19) functional test cases pass with 100% model coverage!
Gaining Confidence in our Design

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

Confidence vs. Effort / Time
Model Advisor – Model Standards Checking

Model Advisor Report - Step_02_fuelsys.slx

Run Summary
- Pass: 0
- Fail: 11
- Not Run: 0
- Total: 12

Modeling Standards for MAAB
- Naming Conventions

- Check file names
  - Identify file names with incorrect characters or formatting.
  - See Also
    - MathWorks Automotive Advisory Board Guideline: ar_0001

- Check folder names
  - Identify folders using incorrect characters and formatting.
  - See Also
    - MathWorks Automotive Advisory Board Guideline: ar_0002
Gaining Confidence in our Design

- Ad-hoc testing
- Design error detection
- Functional & structural tests
- Modeling standards
- Model & code equivalence checks
Equivalence Testing:
Model vs SIL or PIL Mode Testing

Model Testing

SIL or PIL Mode Testing

Coverage $\rightarrow 100$

Model used for production code generation

Embedded Coder

Generated C code

Target compiler and linker

Object code

Execution

Signal comparison

Result vectors ($o_{base}(t)$)

Test vectors ($i(t)$)

Result vectors ($o_{code}(t)$)
Code Generation with Model-to-Code Traceability
Code Generation with Model-to-Code Traceability
Code Equivalence Check Results:  
Model vs Code
Code Equivalence Check Results: Model vs Code
# Code Equivalence Check Results: Model vs Code

## Code Coverage

<table>
<thead>
<tr>
<th>File Contents/Complexity</th>
<th>D1</th>
<th>Cumulative</th>
</tr>
</thead>
<tbody>
<tr>
<td>CruiseControl</td>
<td>22.97%</td>
<td>98%</td>
</tr>
<tr>
<td>CruiseControl_Init</td>
<td>1</td>
<td>--</td>
</tr>
<tr>
<td>CruiseControl</td>
<td>20.97%</td>
<td>98%</td>
</tr>
<tr>
<td>CruiseControl_initialize</td>
<td>1</td>
<td>--</td>
</tr>
</tbody>
</table>

```c
47 /* Output and update for referenced model: 'CruiseControl' */
48 void CruiseControl(const boolean T rtu_CruiseOnOff, const boolean T rtu_Brake,
49   const uint8_T rtu_Speed, const boolean T rtu_CoastSetSw,
50   const boolean T rtu_AccelResSw, boolean T rty_engaged,
51   uint8_T rty_tspeed)
52 {
53   /* Chart: '<Root>/Compute_target_speed' */
54   /* Gateway: Compute target speed */
55   if (CruiseControl_HW.temporalCounter_i1 < MAX_uint32_T) {
```

### Decisions analyzed:

<table>
<thead>
<tr>
<th>Condition</th>
<th>#1</th>
<th>#2</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>CruiseControl_HW.temporalCounter_i1 &lt; MAX_uint32_T</td>
<td>0/399</td>
<td>0</td>
<td>0/420</td>
</tr>
<tr>
<td>true</td>
<td>399/399</td>
<td>0</td>
<td>420/420</td>
</tr>
</tbody>
</table>
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
- Equivalence test performed in Simulink Test, including test execution, evaluation and presentation of the results
- Compared 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 Integration Analysis

ECU

System Inputs

ECU system

1. Cruise Control Module (MBD)

2. Fuel Rate Control Module

Shift Logic Control Module

Outputs

Legacy code
Code Integration Analysis

ECU

Inputs
- Cruise_onoff
- Brake
- Speed
- Coast set
- Accel reset
- EGO Sensor
- MAP Sensor

Outputs
- Gear
- Engaged
- Target speed
- Fuel Rate

ECU system

Cruise Control Module (MBD)

Fuel Rate Control Module

Shift Logic Control Module

Legacy code

Inputs and outputs are connected to the ECU system, with inputs including Cruise_onoff, Brake, Speed, Coast set, Accel reset, EGO Sensor, and MAP Sensor. Outputs include Gear, Engaged, Target speed, and Fuel Rate.
Finding Dead Code During Integration

**Inputs**
- Cruise_onoff
- Brake
- Speed
- Coast set
- Accel reset
- EGO Sensor
- MAP Sensor

**Outputs**
- Gear
- Engaged
- Target speed
- Fuel Rate

**Inputs**
- Inaccurate scaling for speed

**ECU System**
- Cruise Control Module (MBD)
- Fuel Rate Control Module
- Shift Logic Control Module

**Legend**
- Dead code
- Legacy code
Target speed parameter propagated to “Cruise_ctrl.c” [0 … 40]

Maximum target speed = 90

Dead code
Root Cause for Dead Code: Speed Sensor Input Hand Code

Changing analog-to-digital converter from 14 to 12-bit results in dead code

MASK – accounts for scaling down for new ADC from 14-bit to 12-bit

CONV_FACTOR – accounts for translating sensor input counts to mph

Overlooked changing CONV_FACTOR for new ADC
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;
    }
}
Polyspace 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++;
        }
    }

    i = get_bus_status();

    if (i >= 0) {
        *(p = i) = 10;
    }
}
```

**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
Gaining Confidence in our Design

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

Effort / Time

Confidence
Conclusion: Model-Based Design Verification Workflow

Model Verification
Discover design errors at design time

- Module and integration testing at the model level
- Review and static analysis at the model level

Textual requirements → Executable specification → Modeling → Model used for production code generation

Code Verification
Gain confidence in the generated code

- Equivalence testing
- Prevention of unintended functionality

Model used for production code generation → Generated C code → Object code

Workflow approved by TÜV SÜD for development of safety-critical software in accordance with ISO 26262 (automotive), IEC 61508 (industrial), EN 50128 (railway), IEC 62304 (medical devices)
Conclusion

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

Model-Based Design enables:

1. Early testing to increase confidence in your design
2. Delivery of higher quality software throughout the workflow
Change the world by

Accelerating the pace
of discovery, innovation, development, and learning

in engineering and science