Increasing Embedded Software Confidence
Model and Code Verification

MATLAB EXPO 2016

Daniel Martins
Application Engineer – MathWorks
Daniel.martins@mathworks.fr
What is the Cost of Software Failure…

Ariane 5

$7,500,000,000,000

Rocket & payload lost
What is the Cost of Software Failure...

Therac-25

6 Casualties
due to radiation overdose
Where do you want to discover and fix errors?

Pre sales

Model

Vehicle Testing

Generated Code

Post sales

In Service
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

70 km/h
Application: Cruise Control

System Inputs → ECU system → Outputs

1. Cruise Control Module (MBD)
2. Legacy code

Fuel Rate Control Module
Shift Logic Control Module
Application: Cruise Control

ECU

System Inputs

ECU system

Cruise Control Module (MBD)

Legacy code

Fuel Rate Control Module

Shift Logic Control Module

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

Confidence

Ad-hoc testing

Effort / Time
Ad-hoc Tests

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

- Ad-hoc testing
- Design error detection
Finding Design Errors: Dead Logic

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

Results:
- Generate detailed analysis report
- Open harness model
Finding Unintended Behavior

- **Dead logic** due to “uint8” operation on `incdec/holdrate*10`

  ```matlab
  debug>> incdec
  incdec = 
  1
  debug>> holdrate
  holdrate = 
  5
  debug>> incdec/holdrate
  ans = 
  0
  ```

- **Fix** change the order of operation `10*incdec/holdrate`

  ```matlab
  debug>> class(incdec)
  ans = 
  uint8
  debug>> class(holdrate)
  ans = 
  uint8
  ```

Condition can never be false
Gaining Confidence in our Design

Ad-hoc testing
Design error detection
Functional & structural tests
Simulation Testing Workflow

**Requirements**

Did we meet requirements?

**Design**

Did we completely test our model?

**Functional**

Review functional behavior

**Structural**

Structural coverage report

- **Report Generated by Test Manager**
  - Title: LandingGearControl-Regression Tests
  - Author: Janice Johnson
  - Date: 20-Feb-2019 19:28:32
  - Test Environment
  - Test Manager

- **Structural Coverage Report**
  - Table showing coverage metrics for different components of the model.
Requirements Based Functional Testing with Coverage Analysis
Did We Completely Test our Model?

Potential causes of less than 100% coverage:
- Missing requirements
- Over-specified design
- Design errors
- Missing tests

Model Coverage Analysis

Summary

<table>
<thead>
<tr>
<th>Model Hierarchy/Complexity</th>
<th>Test 1</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>D1</td>
<td>C1</td>
<td>MCDC</td>
</tr>
<tr>
<td>1. CruiseControl_IntCalc</td>
<td>35 89%</td>
<td>90%</td>
<td>81%</td>
</tr>
<tr>
<td>2. ComputeTargetSpeed</td>
<td>34 89%</td>
<td>90%</td>
<td>81%</td>
</tr>
<tr>
<td>3. SF: ComputeTargetSpeed</td>
<td>33 89%</td>
<td>90%</td>
<td>81%</td>
</tr>
<tr>
<td>4. SF: CRUISE</td>
<td>30 88%</td>
<td>90%</td>
<td>81%</td>
</tr>
<tr>
<td>5. SF: ON</td>
<td>17 80%</td>
<td>80%</td>
<td>60%</td>
</tr>
</tbody>
</table>
Gaining Confidence in our Design

- Ad-hoc testing
- Design error detection
- Functional & structural tests
- Modeling standards
Model Advisor – Model Standards Checking

1. Check use of Switch blocks

2. MathWorks Automotive Advisory Board Guideline: jg_0141

See Also
- MathWorks Automotive Advisory Board Guideline: ar_0002

Check Switch block parameters

Description
- The switch condition, input 2, must be a Boolean value.
- The block parameter, Criteria for passing first input, should be set to \( u_2 = 0 \).

Correct

![Diagram of Switch block parameters with correct configuration]
Gaining Confidence in our Design

- Ad-hoc testing
- Design error detection
- Functional & structural tests
- Modeling standards
- Model & code equivalence checks
2. Functional Requirements

2.1. Disabled (off) during start-up
Initial state of cruise control system shall be disabled.

2.2. Not engaged with enabling (on)
The cruise control system shall not be initially engaged with enabling.

2.3. Disengaged (not active) when disabled (off)
The cruise control system shall disengage with disabling.

2.4. Initial transition from disengaged (inactive) to engaged (active) only with “Set Speed” input after enabling (on)
The cruise control shall only transition to engaged (active) the first time after the system has been enabled with a “Set Speed” input. Target speed will be set to current vehicle speed.

2.5. “Resume” input ignored until the initial transition from disengaged (inactive) to engaged (active)
Equivalence Testing: Model vs SIL or PIL Mode Testing

Coverage → 100%

Model Testing

- Model used for production code generation
- Simulation
- Result vectors (base line) \( \sigma_{\text{test}}(t) \)

SIL or PIL Mode Testing

- Test vectors \( i(t) \)
- Embedded Coder
- Generated C code
- Target compiler and linker
- Object code
- Execution
- Result vectors \( o_{\text{code}}(t) \)
- Signal comparison
Code Equivalence Check Results: Model vs Code

File Contents/Complexity

<table>
<thead>
<tr>
<th>File Content/Complexity</th>
<th>DI</th>
<th>Cl</th>
<th>MCDC</th>
<th>Stmt</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. CruiseControl.c</td>
<td>22.97%</td>
<td>98%</td>
<td>96%</td>
<td>100%</td>
</tr>
<tr>
<td>2. CruiseControl_Init</td>
<td>1</td>
<td>--</td>
<td>--</td>
<td>100%</td>
</tr>
<tr>
<td>3. CruiseControl</td>
<td>20.97%</td>
<td>98%</td>
<td>96%</td>
<td>100%</td>
</tr>
<tr>
<td>4. CruiseControl_initialize</td>
<td>1</td>
<td>--</td>
<td>--</td>
<td>100%</td>
</tr>
</tbody>
</table>

Code Coverage

Decisions analyzed:

<table>
<thead>
<tr>
<th>Decision</th>
<th>#1</th>
<th>#2</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>false</td>
<td>0/399</td>
<td>0%</td>
<td>0/420</td>
</tr>
<tr>
<td>true</td>
<td>399/399</td>
<td>--</td>
<td>420/420</td>
</tr>
</tbody>
</table>
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
Code Integration Analysis

ECU

System Inputs

ECU system

1

Cruise Control Module (MBD)

Fuel Rate Control Module

Shift Logic Control Module

Outputs

Legacy code
Code Integration Analysis

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

ECU system

Outputs
- Gear
- Engaged
- Target speed
- Fuel Rate

ECU

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

Legacy code

Inputs and Outputs Diagram
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

ECU system

Cruise Control Module (MBD)

Fuel Rate Control Module

Shift Logic Control Module

Dead code

Inaccurate scaling for speed
Finding Dead Code with Polyspace

Target speed parameter propagated to “Cruise_ctrl.c”
[0 … 40]

Maximum target speed = 90

Dead code

```c
/* Entry 'STANDBY': '<S5>:52' */
*rty_Engaged = false;
} else if (rtu_Speed > maxtspeed)
/* Transition: '<S5>:55' */
/* Exit Internal 'ON': '<S5>:54' */
localDW->is_ON = IN_NO_ACTIVE_CHILD;
localDW->is_CRUISE = IN_STANDBY;

/* Entry 'STANDBY': '<S5>:52' */
*rty_Engaged = false;
} else if (rtu_Speed < mntspeed) {
/* Transition: '<S5>:113' */
```
Polyspace Code Analysis

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

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 >= 8) {
        *(*(i - i) + 10) = 10;
    }
}
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

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)
Key Takeaway

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