Increasing Embedded Software Confidence
Model and Code Verification
What is the Cost of Software Failure…

Ariane 5

$7,500,000,000,000

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

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

*Pre sales*
- Model
- Generated Code

*Post sales*
- Vehicle Testing
- In Service
Using Model-Based Design

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

Confidence

Effort / Time

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

Cruise Control Module (MBD)

Fuel Rate Control Module

Shift Logic Control Module

Legacy code

Outputs
Application: Cruise Control

ECU

System Inputs

ECU system

Cruise Control Module

Fuel Rate Control Module

Shift Logic Control Module

Legacy code

Outputs
Application: Cruise Control

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

**Outputs**
- Engaged
- Target speed

Cruise Control Module (MBD)
Gaining Confidence in our Design

Confidence vs. Effort / Time

- Ad-hoc testing
Ad-hoc Tests

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

Confidence

Ad-hoc testing

Design error detection

Effort / Time
Finding Design Errors: Dead Logic

- Design Error Detection: Dead logic
- Results:
  - 2/70 objectives are dead logic.
  - 68/70 objectives are active logic.

Compute target speed.Cruise.ON."
[after(incdec/holdrate... "...
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 \( \text{incdec/holdrate} \times 10 \)

- **Fix** change the order of operation \( 10 \times \text{incdec/holdrate} \)

Condition can never be false
Gaining Confidence in our Design

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

Did we meet requirements?

Review functional behavior

Did we completely test our model?

Structural coverage report

Requirements

Design

Functional

Structural
Requirements Based Functional Testing with Coverage Analysis
Did We Completely Test our Model?

Model Coverage Analysis

Summary

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

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

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

Effort / Time vs Confidence
Model Advisor – Model Standards Checking

1. Check use of Switch blocks

- MathWorks Automotive Advisory Board Guideline: ar_0002

2. MathWorks Automotive Advisory Board Guideline: jc_0141

- Use of the Switch block

- Check use of Switch blocks

Search Documentation

- Simulink
- Modeling Guidelines
- MATLAB Control Algorithm Modeling
- Simulink

Description

- The switch condition, input 2, must be a Boolean value.
- The block parameter, Criteria for passing first input, should be set to u2=0.

Correct

- [Diagram of Switch block]

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
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 $\rightarrow$ 100%

Model Testing

- Model used for production code generation
- Simulation

Test vectors $i(t)$

Embedded Coder

Generated C code

Target compiler and linker

Object code

Execution

Result vectors (base line) $o_{\text{sim}}(t)$

Signal comparison

Result vectors $o_{\text{code}}(t)$
Code Equivalence Check Results: Model vs Code

Code Coverage

<table>
<thead>
<tr>
<th>File Contents/Complexity</th>
<th>DI</th>
<th>Cumulative</th>
</tr>
</thead>
<tbody>
<tr>
<td>CruiseControl_e</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>

Code snippet:

```c
void CruiseControl(const boolean_T *rtu_CruiseOnOff, const boolean_T *rtu_Brake,
                    const uint8_T *rtu_speed, const boolean_T *rtu_CoastSetSw,
                    const boolean_T *rtu_AccelSetSw, const boolean_T *rtY_engaged,
                    uint8_T *rtY_tspeed)
{
  /* Chart: '<Root>/Compute_target_speed' */
  /* Gateway: Compute target speed */
  if (CruiseControl_DW.temporalCounter_i1 < MAX_uint32_T) {
    CruiseControl_DW.Acceleration_prev = rtu_AccelSetSw;
    ...
  }
}
```

Decisions analyzed:

<table>
<thead>
<tr>
<th>Decision</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>false</td>
<td>0/399</td>
</tr>
<tr>
<td>true</td>
<td>399/399</td>
</tr>
</tbody>
</table>

Total: 420/420
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
Code Integration Analysis

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

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

Legacy code

Outputs
- Gear
- Engaged
- Target speed
- 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

ECU system

Cruise Control Module (MBD)

Fuel Rate Control Module

Shift Logic Control Module

Dead code

Legacy 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 < mintspeed) {
/* Transition: '<S5>:113' */
```
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 = 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

Code Verification
Gain confidence in the generated code

- Equivalence testing
- Prevention of unintended functionality

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