Sicher ist sicher:
Standardkonforme Absicherung von High-Integrity Systemen mit Model-Based Design

MATLAB EXPO 2013
Master Class

Dr. Marc Segelken
Senior Application Engineer
Agenda

- Example: How to approach objectives of standards
- Reference Workflow – Example ISO 26262
- Verification Methods and Tools
- Tool Classification and Qualification
Safety Standards for Embedded Systems

- IEC 61508
- ISO 26262
- EN 50128
- DO-178C
- DO-278A
- DO-330
- DO-254
- ...

IEC Certification Kit

DO Qualification Kit
IEC Certification Kit

① Supports tool qualification
② Streamlines ISO 26262 compliant development of embedded systems

www.mathworks.com/products/iec-61508/
## Process Compliance Demonstration

Annotated method tables with suggestions on how to use Model-Based Design processes and tools to apply the methods listed in ISO 26262-6

### Table 9 – Methods for Verification of Software Unit Design and Implementation

<table>
<thead>
<tr>
<th>Methods</th>
<th>ASIL</th>
<th>Applicable Model-Based Design Tools and Processes</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>1a Walkthrough</td>
<td>++ + o o</td>
<td>Simulink Report Generator – Web View, System Design Description (SDD) report</td>
<td>Unit design walkthroughs can be based on a model, a generated Web View, or an SDD report.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Embedded Coder – Code generation report</td>
<td>Code walkthroughs can be based on HTML code generation reports or code generation reports with an integrated Web View of the model.</td>
</tr>
<tr>
<td>1b Inspection</td>
<td>+ ++ ++ +</td>
<td>Simulink Report Generator – Web View, System Design Description (SDD) report</td>
<td>Unit design inspections can be based on a model, a generated Web View, or an SDD report.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Simulink Verification and Validation – Model Advisor checks</td>
<td>Unit design inspections can be supported by ISO 26262.</td>
</tr>
</tbody>
</table>

### Table 10 – Methods for Software Unit Testing

<table>
<thead>
<tr>
<th>Methods</th>
<th>ASIL</th>
<th>Applicable Model-Based Design Tools and Processes</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>1a Requirements-based test</td>
<td>++ + ++ +</td>
<td>Simulink Verification and Validation – Requirements Management Interface (RMI)</td>
<td>RMI can be used to establish bidirectional links between textual requirements and models.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>TEC Certification Kit – Traceability matrix</td>
<td>Generated traceability matrices can be used to document and review existing links between textual requirements, models, and code.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Simulink – Signal Builder block</td>
<td>Signal Builder blocks can be used to create open-loop model tests.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Stateflow – Dynamic test vector charts</td>
<td>Dynamic test vector charts can be used to create closed-loop, reactive model tests.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Simulink Verification and Validation – Component testing capabilities</td>
<td>Component testing capabilities can be used to create model test harnesses. They also enable a requirements pane in the Signal Builder that can be used to link tests with textual requirements.</td>
</tr>
</tbody>
</table>
Reference Workflow

Model-Based Design for ISO 26262

Exemplary verification and validation process for safety-related software created using Model-Based Design
Reference Workflow

Model-Based Design for ISO 26262

Simulation / model testing (Simulink)
Model coverage (Simulink V&V)
Req. Mgmt. Int. (Simulink V&V)

Module and integration testing at the model level

Equivalence testing
Prevention of unintended functionality

Review and static analysis at the model level

Textual requirements
Executable specification
Modeling

Model used for production code generation
Generated C code
Object code

Code generation
Compilation and linking
Test Coverage Analysis for Models

Simulation / model testing (Simulink)
Model coverage (Simulink V&V)
Req. Mgmt. Int. (Simulink V&V)

Module and integration testing at the model level
Review and static analysis at the model level
Equivalence testing
Prevention of unintended functionality

Textual requirements → Executable specification → Model used for production code generation → Generated C code → Object code

Modeling
Code generation
Compilation and linking
Creating links between textual documents and model objects

- Simulation / model testing (Simulink)
- Model coverage (Simulink V&V)
- Req. Mgmt. Int. (Simulink V&V)
Traceability Matrix Generation

Generate traceability information covering requirements, model elements, and generated code

- Use Microsoft Excel to track, and annotate generated traceability matrices for your project
- Fulfill ISO 26262 requirements to document traceability information and to demonstrate absence of unintended functionality (cf. ISO 26262-6, 8.4.6)

```matlab
/* Exported block signals */
real_T INPUT;
real_T OUTPUT;

/* Exported block parameters */
real_T k = 5.0;
OUTPUT = INPUT * k;
```

```
> iec.ExportTraceReport('model_name')
```
ISO 26262 Modeling Guidelines Checks

Model standards checking (Simulink V&V)

Module and integration testing at the model level

Review and static analysis at the model level

Equivalence testing

Prevention of unintended functionality

Textual requirements

Executable specification

Modeling

Model used for production code generation

Generated C code

Object code

Code generation

Compilation and linking
ISO 26262 Modeling Guidelines Checks

Model Advisor checks to facilitate ISO 26262-6 objectives at the model level
Code Generation Verification in the context of ISO 26262
Code Generation Verification in the context of ISO 26262

- Support for software unit and integration testing objectives in ISO 26262-6

Table 12 — Methods for software unit testing

<table>
<thead>
<tr>
<th>Methods</th>
<th>ASIL A</th>
<th>ASIL B</th>
<th>ASIL C</th>
<th>ASIL D</th>
</tr>
</thead>
<tbody>
<tr>
<td>1a Requirement-based test</td>
<td>++</td>
<td>++</td>
<td>++</td>
<td>++</td>
</tr>
<tr>
<td>1b Interface test</td>
<td>++</td>
<td>++</td>
<td>++</td>
<td>++</td>
</tr>
<tr>
<td>1c Fault injection testa</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>++</td>
</tr>
<tr>
<td>2d Resource usage testb</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>++</td>
</tr>
<tr>
<td>3e Back-to-back test between model and code, if applicablec</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>++</td>
</tr>
</tbody>
</table>

* a This includes injection of arbitrary faults in order to test safety mechanisms (e.g., by computing values of variables).

b Some aspects of the resource usage test can only be evaluated properly when the software unit tests are executed on the target hardware or if the emulator for the target processor supports resource usage tests.

c This method requires a model that can simulate the functionality of the software units. Here, the model and code are stimulated in the same way and results compared with each other.

Table 15 — Methods for software integration testing

<table>
<thead>
<tr>
<th>Methods</th>
<th>ASIL A</th>
<th>ASIL B</th>
<th>ASIL C</th>
<th>ASIL D</th>
</tr>
</thead>
<tbody>
<tr>
<td>1a Requirements-based test</td>
<td>++</td>
<td>++</td>
<td>++</td>
<td>++</td>
</tr>
<tr>
<td>1b External interface test</td>
<td>++</td>
<td>++</td>
<td>++</td>
<td>++</td>
</tr>
<tr>
<td>1c Fault injection testc</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>++</td>
</tr>
<tr>
<td>2d Resource usage testc</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>++</td>
</tr>
<tr>
<td>3e Back-to-back test between model and code, if applicabled</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>++</td>
</tr>
</tbody>
</table>

* a This includes injection of arbitrary faults in order to test safety mechanisms (e.g., by computing values of variables).

b To ensure the fulfilment of requirements influenced by the hardware architectural design with sufficient tolerance, properties such as average and maximum processor performance, minimum or maximum execution times, storage usage (e.g., RAM for stack and heap, ROM for program and data) and the bandwidth of communication links (e.g., data busses) have to be determined.

c Some aspects of the resource usage test can only be evaluated properly when the software integration tests are executed on the target hardware or if the emulator for the target processor supports resource usage tests.

d This method requires a model that can simulate the functionality of the software components. Here, the model and code are simulated in the same way and results compared with each other.
Reference Workflow

Model-Based Design for ISO 26262

Model standards checking (Simulink V&V)

Simulation / model testing (Simulink)
Model coverage (Simulink V&V)
Req. Mgmt. Int. (Simulink V&V)

Module and integration testing at the model level

Review and static analysis at the model level

PIL test (Embedded Coder)

Traceability matrix analysis (IEC Certification Kit) or model vs. code coverage (third-party tool)

Equivalence testing

Prevention of unintended functionality

Textual requirements

Executable specification

Model used for production code generation

Generated C code

Object code

Modeling

Simulink/Stateflow

Code generation

Compilation and linking

Embedded Coder

Third-party tool
Advanced Reference Workflow
Additional Best Practices

Model standards checking (Simulink V&V)
Property Proving (Simulink Design Verifier)

Simulation / model testing (Simulink)
Model coverage (Simulink V&V)
Req. Mgmt. Int. (Simulink V&V)

Module and integration testing at the model level

Review and static analysis at the model level

PIL test (Embedded Coder)
Test generation (Simulink Design Verifier)
Run-time error detection (Polyspace products)

Traceability matrix analysis (IEC Certification Kit) or model vs. code coverage (third-party tool)
MISRA-C checking (Polyspace products)

Equivalence testing
Prevention of unintended functionality

Model used for production code generation
Generated C code
Object code

Textual requirements → Executable specification → ...

Modeling
Simulink/Stateflow

Code generation
Compilation and linking
Embedded Coder
Third-party tool
## Reference Workflow - **Compliance Demonstration**

### Templates

<table>
<thead>
<tr>
<th>Checklist 1: Design Verification</th>
<th>Checklist 2: Code Verification</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Technique / Measure</strong></td>
<td><strong>Technique / Measure</strong></td>
</tr>
<tr>
<td>Model review (See &quot;Review and Static Analysis at the Model Level&quot;)</td>
<td>Equivalence test vector generation (See &quot;Equivalence Test Vector Generation&quot;)</td>
</tr>
<tr>
<td>Adherence to modeling standard (See &quot;Review and Static Analysis at the Model Level&quot;)</td>
<td>Equivalence test execution (See &quot;Equivalence Test Vector Generation&quot;)</td>
</tr>
<tr>
<td>Static analysis at the model level (if applicable) (See &quot;Review and Static Analysis at the Model Level&quot;)</td>
<td>Signal comparison (See &quot;Signal Comparison&quot;)</td>
</tr>
<tr>
<td>Supporting activities (See &quot;Review and Static Analysis at the Model Level&quot;)</td>
<td><strong>Associated Requirements</strong></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>Module and integration testing at the model level</strong></th>
<th><strong>Equivalence testing</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Review and static analysis at the model level</strong></td>
<td><strong>Prevention of unintended functionality</strong></td>
</tr>
</tbody>
</table>

**Textual requirements** → **Executable specification** → **Modeling** → **Model used for production code generation** → **Generated C code** → **Object code** → **Code generation** → **Compilation and linking**
IEC Certification Kit

1. Supports tool qualification
2. Streamlines ISO 26262 compliant development of embedded systems

www.mathworks.com/products/iec-61508/
I. Tool Classification

- Tool use cases
  - UC 1..n
  - TI 2
  - TI 1

- Tool impact
  - TI 2

- Tool error detection
  - TD 3
  - TD 2
  - TD 1

- Tool confidence level
  - High
  - Medium
  - Increasing qualification requirements

II. Tool Qualification

- ASIL
- Qualification methods for TCL3
- Qualification methods for TCL2
- Qualification not required

ISO 26262 Tool Qualification Approach - Details
Qualification of MathWorks Tools

Tool qualification may involve multiple parties

- **Tool user**
  - Responsible for final tool qualification in the context of the application

- **Tool vendor**
  - Conducts generic pre-classification and pre-qualification based on reference use cases / reference workflow
  - Supports / streamlines user’s activities by providing a tool qualification kit

- **3rd party assessor** (optional)
  - Provides independent assessment of reference workflow and pre-qualification artifacts

ISO 26262 Tool Qualification Kit

TÜV SÜD Independent Assessment
Independent Assessment by TÜV SÜD

Example

Certificate

Assessment Report

Report to the
Certificate
Z10 11 01 67052 008
Software Tools for Safety Related Development
Simulink® Verification and Validation™
Simulink® Design Verifier™

Manufacturer:
The MathWorks, Inc.
3 Apple Hill Drive
Natick, MA, 01760-2098
USA

Report No.: MN85534C
Revision 1.1 dated Jan. 24, 2011

5.2 Usage considerations for development processes which need to comply with IEC 61508, ISO 26262, EN 50128, or derivative standards

The capabilities of Simulink® Verification and Validation™ and Simulink® Design Verifier™ listed in sections 2.1.2 and 2.2.2 respectively are certified for use in development processes which need to comply with IEC 61508, ISO 26262, EN 50128, or derivative standards. The two verification tools allow the automation of core verification and validation activities for Simulink models and generated code.
Qualification of Model-Based Design Tools

- Embedded Coder, Simulink Design Verifier, Simulink Verification and Validation, and Polyspace are pre-qualified for all ASILs according to ISO 26262

Note: Embedded Coder, Simulink Design Verifier, Simulink Verification and Validation, and Polyspace products were not developed using certified processes.
Simulink Code Inspector
Automate DO-178C Code Reviews

Independently verify that Embedded Coder generated code traces to and complies with low-level requirements

- Demonstrate that model and source code match structurally
- Provide model ⇔ code traceability data
- Eliminate/reduce manual code reviews for DO-178C software
- Same certification credits as qualified code generator
Summary

- Model-Based Design offers many Verification and Validation methods to address objectives of standards.

- Complete Reference Workflows are guiding through the development process.

- Certification and Qualification Kits easing Tool Qualification process.
ISO 26262 Process Deployment Advisory Service

MathWorks Consulting services to quickly adopt Model-Based Design for ISO 26262

- Objectives
  - Identify gaps in current processes
  - Provide a roadmap to an optimized ISO 26262 process
  - Assist with deployment of that roadmap
  - Educate on the ISO 26262 standard
  - Help with software tool qualification

www.mathworks.com/services/consulting/areas/iso26262-process-deployment.html