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

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

Decision coverage
Condition coverage
MC/DC
Lookup table coverage
Signal range coverage

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

- Creating links between textual documents and model objects

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
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)

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

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

```
<table>
<thead>
<tr>
<th>Model Object Type</th>
<th>Model Object Path</th>
<th>Model Object Subsystem</th>
<th>Model Object SID</th>
<th>Model Object Optimized</th>
<th>Model Optimization Rationale</th>
<th>Code File Name</th>
<th>Code Function</th>
<th>Code Line Number</th>
<th>Code File Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>Inport</td>
<td>model1</td>
<td>model1</td>
<td>model1:1</td>
<td>model1:4</td>
<td></td>
<td>model1.c</td>
<td>Global</td>
<td>22</td>
<td>C:\work\model1_ert_matlab \work\req1.doc</td>
</tr>
<tr>
<td>Inport</td>
<td>model1</td>
<td>model1</td>
<td>model1:1</td>
<td>model1:4</td>
<td></td>
<td>model1.c</td>
<td>model1_step</td>
<td>38</td>
<td>C:\work\model1_ert_matlab \work\req1.doc</td>
</tr>
<tr>
<td>Inport</td>
<td>model1</td>
<td>model1</td>
<td>model1:1</td>
<td>model1:4</td>
<td></td>
<td>model1.c</td>
<td>model1_step</td>
<td>43</td>
<td>C:\work\model1_ert_matlab \work\req1.doc</td>
</tr>
<tr>
<td>Inport</td>
<td>model1</td>
<td>model1</td>
<td>model1:1</td>
<td>model1:4</td>
<td></td>
<td>model1.h</td>
<td>Global</td>
<td>49</td>
<td>C:\work\model1_ert_matlab \work\req1.doc</td>
</tr>
<tr>
<td>Gain</td>
<td>model1</td>
<td>model1</td>
<td>model1:2</td>
<td>model1:2</td>
<td></td>
<td>model1.c</td>
<td>Global</td>
<td>23</td>
<td>C:\work\model1_ert_matlab \work\req1.doc</td>
</tr>
<tr>
<td>Gain</td>
<td>model1</td>
<td>model1</td>
<td>model1:2</td>
<td>model1:2</td>
<td></td>
<td>model1.c</td>
<td>Global</td>
<td>27</td>
<td>C:\work\model1_ert_matlab \work\req1.doc</td>
</tr>
</tbody>
</table>
```
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 → Model used for production code generation → Generated C code → Object code

Modeling

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

PIL test (Embedded Coder)

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
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 test</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>++</td>
</tr>
<tr>
<td>1d Resource usage test</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>++</td>
</tr>
<tr>
<td>1e Back-to-back test between model and code, if applicable</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>++</td>
</tr>
</tbody>
</table>

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

*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.

*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 Requirement-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 test</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>++</td>
</tr>
<tr>
<td>1d Resource usage test</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>++</td>
</tr>
<tr>
<td>1e Back-to-back test between model and code, if applicable</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>++</td>
</tr>
</tbody>
</table>

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

*To ensure the fulfillment 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.

*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.

*This method requires a model that can simulate the functionality of the software components. Here, the model and code are stimulated 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

Equivalence testing

Prevention of unintended functionality

Review and static analysis at the model level

Model used for production code generation

Generated C code

Object code

Textual requirements → Executable specification → ...

Modeling

Simulink/Stateflow

PIL test (Embedded Coder)

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

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

**Templates**

### Checklist 1: Design Verification

<table>
<thead>
<tr>
<th>Technique / Measure</th>
<th>Associated Requirements</th>
<th>Used / Used to a limited degree / Not used</th>
<th>Interpretation in the application, Evidence</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 Model review (See “Review and Static Analysis at the Model Level”)</td>
<td>▶ Inclusion of all model components</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
| 2 Adherence to modeling standard (See “Review and Static Analysis at the Model Level”) | ▶ Designation of a modeling standard
   ▶ Ensure the modeling standard is suitable for use
   ▶ Restrictions on modeling constructs used for production code generation
   ▶ Evidence for using the modeling standard                                                                 |                                           |                                            |
| 3 Static analysis at the model level (if applicable) (See “Review and Static Analysis at the Model Level”) | ▶ Evidence for using static analysis                                                    |                                           |                                            |
| 4 Supporting activities (See “Review and Static Analysis at the Model Level”) | ▶ Documentation of the results of reviews and analyses
   ▶ Corrective action on failure of reviews and analyses                                                                 |                                           |                                            |

### Checklist 2: Code Verification

<table>
<thead>
<tr>
<th>Technique / Measure</th>
<th>Associated Requirements</th>
<th>Used / Used to a limited degree / Not used</th>
<th>Interpretation in the application, Evidence</th>
</tr>
</thead>
</table>
| 7 Equivalence test vector generation (See “Equivalence Test Vector Generation”) | ▶ Equivalence test vector generation
   ▶ Simulation of the model used for production code generation
   ▶ Simulation of the executable derived from the generated code
   ▶ Test execution in a target environment or analysis of the differences between testing and the target environment
   ▶ Assessment of model parts used for simulation but not for code generation                                                                 |                                           |                                            |
| 8 Equivalence test execution (See “Equivalence Test Vector Generation”) | ▶ Equivalence test execution
   ▶ Simulation of the model used for production code generation
   ▶ Simulation of the executable derived from the generated code
   ▶ Test execution in a target environment or analysis of the differences between testing and the target environment
   ▶ Assessment of model parts used for simulation but not for code generation                                                                 |                                           |                                            |
| 9 Signal comparators (See “Signal Comparisons”) | ▶ Equivalence test execution
   ▶ Simulation of the model used for production code generation
   ▶ Simulation of the executable derived from the generated code
   ▶ Test execution in a target environment or analysis of the differences between testing and the target environment
   ▶ Assessment of model parts used for simulation but not for code generation                                                                 |                                           |                                            |

---

**Module and integration testing at the model level**

**Review and static analysis at the model level**

**Equivalence testing**

**Prevention of unintended functionality**

**Model used for production code generation**

**Generated C code**

**Object code**

**Textual requirements**

**Executable specification**

**Modeling**

**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

<table>
<thead>
<tr>
<th>Tool use cases</th>
<th>Tool impact</th>
<th>Tool error detection</th>
<th>Tool confidence level</th>
</tr>
</thead>
<tbody>
<tr>
<td>UC 1..n</td>
<td>TI 2</td>
<td>TD 3</td>
<td>TCL 3</td>
</tr>
<tr>
<td></td>
<td>TI 1</td>
<td>TD 2</td>
<td>TCL 2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>TD 1</td>
<td>TCL 1</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>High</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Medium</td>
</tr>
</tbody>
</table>

II. Tool Qualification

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

Increasing qualification requirements
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
Independent Assessment by TÜV SÜD

Example

Certificate

Assessment Report

Report to the
Certificate
Z10 11 01 67052 008

Simulink® Verification and Validation™
Simulink® Design Verifier™

Manufacturer:
The MathWorks, Inc.
3 Apple Hill Drive
Natick, MA, 01760-2998
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

[Website](http://www.mathworks.com/services/consulting/areas/iso26262-process-deployment.html)