2009-01-0271

A Verification and Validation Workflow for IEC 61508 Applications

Mirko Conrad
The MathWorks, Inc., Natick, MA, USA

Guido Sandmann
The MathWorks GmbH, Munich, Germany

ABSTRACT

Because of its ability to address software complexity and productivity challenges, Model-Based Design with production code generation has been extensively used throughout the automotive software engineering community. More recently, engineers have begun to focus on compliance with external standards such as IEC 61508 and the use of Model-Based Design.

For in-vehicle applications, the standard applied is typically IEC 61508-3. To demonstrate standard compliance, the objectives and recommendations outlined in IEC 61508-3 have to be mapped onto Model-Based Design processes and tools.

This paper discusses a verification and validation workflow for developing in-vehicle software components that need to meet IEC 61508 using Model-Based Design.

INTRODUCTION

In the last decades, in-vehicle software has become increasingly complex. The amount of functionality, which has to be calculated by each control unit, and the communication between control units, have risen considerably.

To meet these challenges, the development process plays a significant role. Model-Based Design for automotive control units along the V-model is gaining widespread acceptance in applications, because it offers a series of advantages. Modeling facilitates the communication between OEMs and suppliers, and also between engineers in their projects.

At the center of Model-Based Design is an executable model representing the embedded software component to be developed. The model serves as the primary representation throughout multiple phases of the development process. An initial executable model (executable specification) is refined and augmented until it becomes a blueprint for the final implementation. In addition, executable models can be used for various verification and validation activities.

Because of its ability to address complexity and productivity challenges, Model-Based Design has been extensively used throughout the software engineering community. But more and more, projects must comply with standards, because modern ECUs and their application software directly interact with systems such as brakes and steering.

OEMs and suppliers have recently begun to consider Model-Based Design for the development of embedded software for applications that need to meet the IEC 61508 standard. Examples include application software components of the electromechanical APA steering system [JSB+08] for the Volkswagen Tiguan [FMC08].

For automotive in-vehicle applications, IEC 61508-3 is often considered as state-of-the art within the industry. To demonstrate compliance with the standard, the objectives and recommendations outlined in IEC 61508-3 need to be mapped onto Model-Based Design processes and tools.

The Engineering Meetings Board has approved this paper for publication. It has successfully completed SAE's peer review process under the supervision of the session organizer. This process requires a minimum of three (3) reviews by industry experts.

ISSN 0148-7191

SAE Customer Service: Tel: 877-606-7323 (inside USA and Canada)
Tel: 724-776-4970 (outside USA)
Fax: 724-776-0790
Email: CustomerService@sae.org
SAE Web Address: http://www.sae.org

Printed in USA

The remainder of this paper discusses a verification and validation workflow for applications that need to meet IEC 61508 using Model-Based Design.

A WORKFLOW FOR APPLICATION-SPECIFIC VERIFICATION AND VALIDATION OF MODELS AND GENERATED CODE

As other standards do, IEC 61508-3 calls for **application-specific verification and validation** regardless of the tool chain and the development paradigm used.

For applications implemented with Model-Based Design, the following two questions apply to application-specific verification and validation:

1. Does the model correctly implement its (textual) requirements?
2. Does the object code that will be deployed in the ECU correctly implement the model's behavior?

To facilitate the seamless use of Model-Based Design in the context of IEC 61508, a reference workflow that describes the verification and validation activities necessary for IEC 61508-3 compliance has been developed.

Following the above suggested division of the verification and validation task for applications developed using Model-Based Design and production code generation, the workflow would be divided into the following two steps (cf. [Ald01]):

1. Demonstrate that the model is correct and meets all requirements
2. Show that the generated code is equivalent to the model

The first step is **design verification**, which combines suitable verification and validation techniques at the model level. The second step is **code verification**.

VERIFICATION AND VALIDATION AT THE MODEL LEVEL (DESIGN VERIFICATION) - The goal of design verification is to gain confidence in the model, which is then used for production code generation. This step takes place at the model level, i.e., before the code is generated.

Following the spirit of IEC 61508-3, the design verification part of the reference workflow therefore comprises a combination of reviews, static analyses, and comprehensive functional testing activities at the model level [Con08]. These activities together help provide confidence that the design satisfies the associated requirements. The result is a **golden model**, i.e., a sufficiently validated and verified model that implements the requirements and does not contain any unintended functionality.

Requirements for design verification can be derived from IEC 61508-3 clauses 7.4.6 (code implementation), 7.4.7 (software module testing), and 7.4.8 (software integration testing).

Reviews and Static Analyses at the Model Level - Model subsystems considered as modules (model components) should be reviewed. If feasible, manual **model reviews** should be supported by automated **static analyses** of the model.

Modeling guidelines should be used, and adherence with the guidelines should to be assessed. Modeling constructs that are not suitable or not recommended for production code generation should not be used.

Tool support: Model reviews can be facilitated by using reports or Web views generated with Simulink Report Generator. Adherence to a modeling guideline can be partially enforced by using predefined or customized modeling standard checks in Model Advisor [Beg07].

Module and Integration Testing on the Model Level - Model components should be functionally tested using systematically derived test vectors. The objective of this **module testing** is to demonstrate that each model component performs its intended function and does not perform any unintended functions.

As module testing is completed, module **integration testing** should be performed with predefined test vectors, i.e., the model integration stages should be tested in accordance with the specified integration tests. These tests should show that all model modules and model subsystems interact correctly to perform their intended function and do not perform unintended functions.

Tool support: Simulink Verification and Validation supports various facets of model testing.

VERIFICATION AND VALIDATION AT THE CODE LEVEL (CODE VERIFICATION) - We use **translation validation through systematic testing (translation testing)** to demonstrate that the execution semantics of the model are being preserved during code generation, compilation, and linking.

Technically speaking, we are using numerical **equivalence testing** between the model used for production code generation and the executable derived from the generated C code, and additional measures to demonstrate the **absence of unintended functionality**.

Figure 1 gives an overview of the proposed translation validation process for generated code.

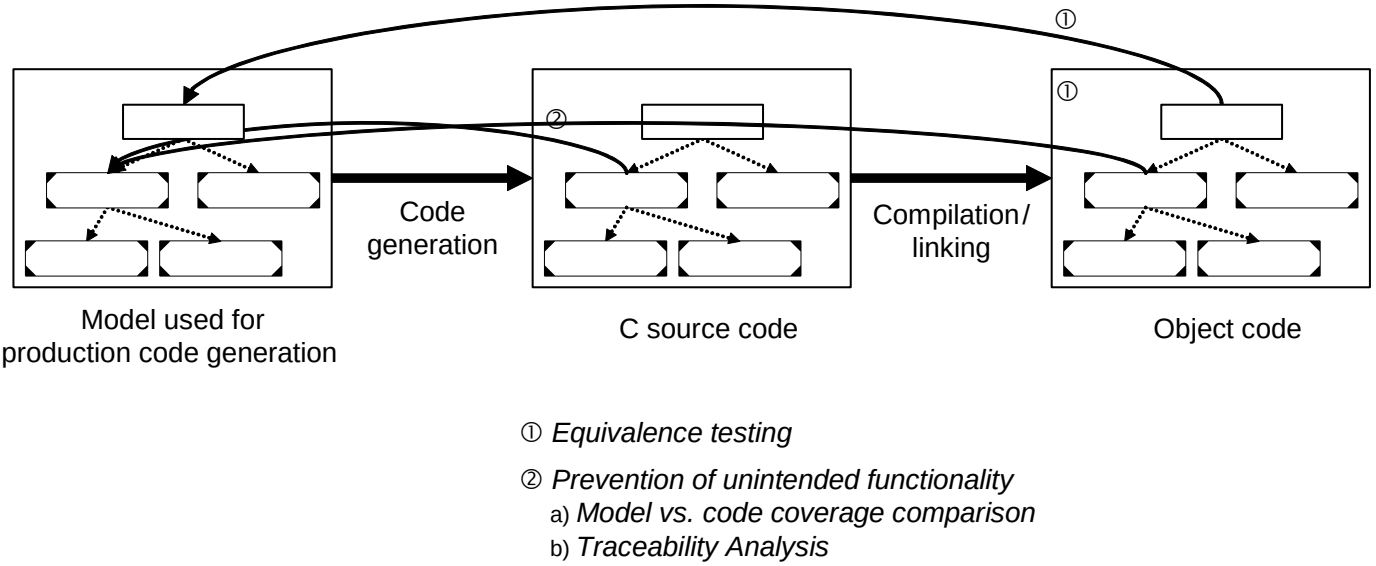

Figure 1: Translation validation process.

Numerical Equivalence Testing - Equivalence testing is performed to demonstrate numerical equivalence between the model and the generated code. Equivalence testing between the model used for production code generation and the resulting object code (also known as comparative testing or back-to-back testing) constitutes a core part of the code verification workflow.

Equivalence testing refers to the stimulation of both the model used for code generation and the object code derived from it through code generation and compilation with identical test vectors. The validity of the translation process, i.e., whether or not the semantics of the model have been preserved during code generation, compilation, and linking, is determined by comparing the system reactions, or result vectors, of the model and the generated code resulting from stimulation with identical timed test vectors *i(t)*. More precisely, the simulation results of the model used for production code generation $o_{SIM}(t)$ are compared with the execution results of the generated and compiled production code $o_{CODE}(t)$.

Figure 2 summarizes the suggested translation validation approach. In-depth discussions of equivalence testing procedures can be found in [SC03, SC05, SCD+07].

Testing for numerical equivalence is unique in that the expected outputs for the test vectors do not have to be provided [Ald01]. This makes equivalence testing well suited to automation.

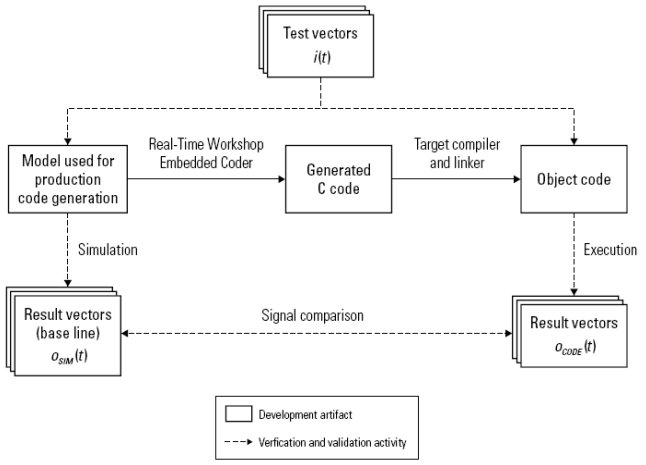

Figure 2: Numerical equivalence testing.

The following subsections provide detailed information on the equivalence testing procedure.

Equivalence Test Vector Generation: A valid translation requires that the execution of the object code exhibit the same observable effects as the simulation of the model for any given set of test vectors. Because complete testing is impossible for complexity reasons, stimuli (test vectors) shall be sufficient to cover the different structural parts of the model.

To assess the model coverage achieved, some **test coverage metric** shall be visible at SIL 2 and above [SS05]. The extent and scope of structural model coverage needs to be increased for the higher SILs.

Tool support: Model coverage analysis can be performed by using the Model Coverage Tool in Simulink Verification and Validation [Simulink Verification and Validation].

Test vectors resulting from requirements-based testing at the model level can be reused for equivalence testing. Figure 3 illustrates equivalence testing of individual and integrated modules[1].

If the coverage achieved with the existing test vectors is not sufficient, additional test vectors should be created.

If full coverage for the selected metric(s) cannot be achieved, the uncovered parts should be assessed and justification for uncovered parts provided. In practice, the set of test vectors can be iteratively extended using model coverage analysis until the mandated level of model coverage has been achieved.

[1] $i_{COMP}(t)$ and $o_{COMP}(t)$ refer to the test vectors and result vectors for model components respectively; $i_{INT}(t)$ and $o_{INT}(t)$ refer to the test vectors and result vectors for the integration stages respectively.

Tool Support: Simulink Design Verifier can be used to create additional test vectors for equivalence testing [Simulink Design Verifier].

Equivalence Test Execution: The test vectors for equivalence testing shall be used to stimulate both the model used for production code generation and the executable derived from the generated code.

The resulting object code shall be tested in an execution environment that corresponds as far as possible to the target environment to which the code will be deployed.

The resulting object code can either be executed on the target processor or on a target-like processor, e.g., by means of a processor-in-the-loop simulation (PIL verification), or simulated by means of an instruction set simulator for the target processor (ISS verification). If feasible, PIL verification is the preferred approach.

If the execution of the resulting object code is not carried out in the target environment, differences between the testing environment and the target environment should be analyzed to make sure that they do not adversely alter the results.

Signal Comparison: After test execution, the result vectors (simulation results) of the model $o_{SIM}(t)$ should be compared with the execution results of the generated code $o_{CODE}(t)$.

The simulation results of the model are used as the baseline. They are matched to the result vectors obtained from executing the object code.

Even if there is a correct translation of a Simulink or Stateflow model into C code, one cannot always expect identical behavior (equality). Possible reasons include the limited precision of floating-point numbers, quantization effects when using fixed-point math, and differences among compilers. For these reasons, the definition of correctness has to be based on sufficiently similar behavior (sufficient similarity).

A suitable signal comparison algorithm should be selected that is able to tolerate differences between the result vectors representing the system responses $o_{SIM}(t)$ and $o_{CODE}(t)$. There is a broad variety of potential comparison algorithms, ranging from simple algorithms, such as absolute difference, to more elaborated ones such as the difference matrix method.

Two result vectors are sufficiently similar if their difference with regard to a given comparison algorithm is less than or equal to a given threshold. The selection of the comparison algorithm and the definition of the threshold value depend on the application under consideration and need to be documented.

Simulink Fixed Point software enables performing bit-true simulations of model portions implemented using

fixed point math to observe the effects of limited range and precision on designs built with Simulink and Stateflow [Simulink Fixed Point]. When used with Real Time Workshop Embedded Coder, Simulink Fixed Point enables pure integer C code to be generated from these model portions. The generated code is in bit-true agreement with the model used for production code generation, ensuring that the implemented design will perform exactly as it did in simulation.

Sufficient similarity serves as a basis for defining functional equivalence used to determine the numerical correctness of the model-to-code translation: A model and the code generated from it are regarded to be functionally equivalent if the simulation of the model and the execution of the executable derived from the generated code lead to sufficiently similar result vectors if both are stimulated with identical test vectors.

Tool support: Comparison algorithms can be implemented in a general-purpose programming or scripting language such as the MATLAB language. MEval [MEval] provides a variety of predefined algorithms for result vector comparison.

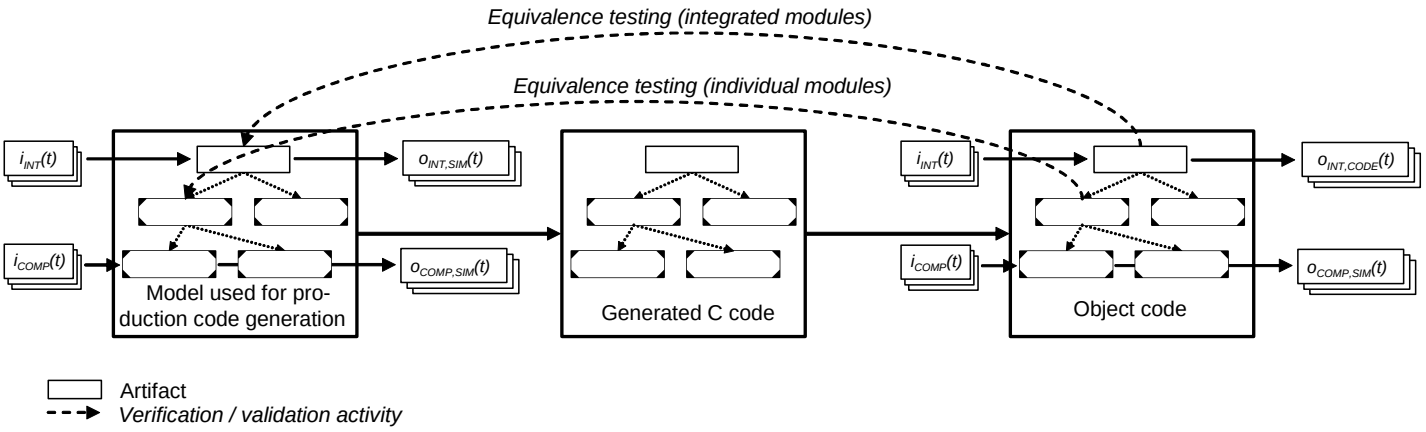

Figure 1: Equivalence testing of individual and integrated modules.

Prevention of Unintended Functionality - The second activity in the code verification process is to demonstrate that the generated C code does not perform any **unintended function**.

Alternative techniques are available to achieve this objective. They serve the purpose of demonstrating structural equivalence between the model and the source code.

One can use at least one of the following measures to demonstrate that the generated C code does not perform any **unintended function**:

- Model vs. code coverage comparison
- Traceability review

Model Versus Code Coverage Comparison: If **model vs. code coverage comparison** is being used, model and code coverage are measured during equivalence testing and compared against each other. Discrepancies with regard to comparable coverage metrics should be assessed.

To be meaningful, structural coverage metrics comparable with each other should be used on the model and code level respectively. According to [BCS+03], decision coverage at the model level and branch coverage (C1) at the code level (which is also sometimes termed decision coverage) can be used in combination.

Discrepancies between model and code coverage with regard to comparable metrics shall be assessed. If the code coverage achieved is less than the model coverage, unintended functionality could have been introduced.

Tool Support: Code coverage information can be derived from the IDE or by applying a standalone code coverage tool to the generated source code.

Traceability Review: A **traceability analysis** of the generated C source code can be performed to ensure that all parts of this code can be traced back to the model used for production code generation. In this case the generated code is subjected to a **limited review** that exclusively focuses on traceability aspects.

Nontraceable code shall be assessed.

Tool Support: Automatically generated Simulink block comments can be used to generate tracing information into the generated code [Real-Time Workshop

Embedded Coder]. The Traceability Report from the Real-Time Workshop Embedded Coder code generation report helps to provide a complete mapping between model blocks and the generated code. It lists eliminated or virtual blocks versus traceable blocks. 'Code-to-block highlighting' generates hyperlinks within the displayed source code to view the blocks or subsystems from which the code was generated. 'Block-to-code highlighting' allows for any block in the model to identify the resulting generated code.

CONCLUSION

Currently, IEC 61508-3 is a relevant standard with respect to software development for embedded in-vehicle applications. It defines requirements and constraints for the software development and quality assurance processes. These requirements apply to both Model-Based Design and traditional software development. However, implementing these requirements within Model-Based Design requires special consideration and creates specific challenges.

In this paper, the authors discuss Model-Based Design workflows and tool considerations to address the objectives of IEC 61508-3.

The proposed workflow for verification and validation of models and generated code could be viewed as an instantiation of the verification and validation requirements outlined in IEC 61508-3 that exploits the advantages of production code generation as an integral part of Model-Based Design.

REFERENCES

1. [Ald01] Aldrich, W., Coverage Analysis for Model-Based Design Tools, TCS 2001.
2. [BCS+03] Baresel, A., Conrad, M.,Sadeghipour, S., Wegener, J., The Interplay Between Model Coverage and Code Coverage, 11. European Int. Conf. on Software Testing, Analysis and Review (EuroSTAR '03), Amsterdam, Netherlands, 2003.
3. [Beg07] Begic, G., Checking Modeling Standards Implementation, The MathWorks News & Notes, June 2007.
4. [Bur04] Burnard, A., Verifying and Validating Automatically Generated Code, Int. Automotive Conference (IAC '04) Stuttgart, Germany, 2004, pp. 71-78.
5. [Con07] Conrad, M., Using Simulink and Real-Time Workshop Embedded Coder for Safety-Critical Automotive Applications, Proc. Workshop Modellbasierte Entwicklung Eingebetteter Systeme III (MBEES'07), Schloß Dagstuhl, Germany, 2007, pp. 41-50.
6. [Con08] Conrad, M., Model-Based Design for IIEC 61508: Towards Translation Validation of Generated Code, Proc. Workshop Automotive Software Engineering: Forschung, Lehre, Industrielle Praxis, colocated with Software Engineering 2008, Munich, February 2008.
7. [EC07] Erkkinen, T. and Conrad, M., Safety-Critical Software Development Using Automatic Production Code Generation, Proc. SAE World Congress 2007, Detroit, USA, 2007.
8. [ECCert] www.mathworks.com/company/pressroom/articles/article18304.html
9. Ekkehard Pofahl, Torsten Sauer, Oliver Busa.
10. [ECAVS] www.mathworks.com/company/pressroom/articles/article17790.html
11. [Edw99] Edwards, P.D., The Use of Automatic Code Generation Tools in the Development of Safety-Related Embedded Systems, Proc. Vehicle Electronic Systems, ERA Report 99-0484, 1999.
12. [Embedded MATLAB] Embedded MATLAB feature page: www.mathworks.com/products/featured/embeddedmatlab
13. [FMC08] Fey, I., Müller, J., Conrad, M., Model-Based Design for Safety-Related Applications, Proc. Convergence 2008, Detroit, MI, USA, Oct. 2008.
14. [HC06] Harmon, R., Hote, C., Automatic Engine Control Code Generation with Integrated Automatic Static Code Verification, International Automotive Conference (IAC '04), Stuttgart, Germany, 2006.
15. [IEC61508-3] IEC 61508-3:1998, Int. Standard Functional Safety of Electrical/Electronic/Programmable Electronic Safety-Related Systems - Part 3: Software Requirements, First edition, 1998.
16. [JSB+08] Thorsten Jablonski, Heiko Schumann, Carsten Busse, Heiko Haussmann, Udo Hallmann, Dirk Dreyer, Frank Schöttler, Die neue elektromechanische Lenkung APA-BS, ATZelektronik 01/2008 Vol. 3 (2008) 01, pp. 30-35.
17. [Model-Based Design] Model-Based Design Web page: www.mathworks.com/applications/controldesign/description
18. [MEval] MEval product page: www.itpower.de/meval_e.html

CONTACT

Dr. Mirko Conrad, Development Manager, Simulink Certification and Standards, The MathWorks, Inc.

Mirko Conrad is a development manager at The MathWorks in Natick, MA, where he leads the Simulink Certification and Standards team.

He has previous automotive experience as a senior research scientist and project manager at Daimler-Benz / DaimlerChrysler.

Mirko holds a Ph.D. in engineering (Dr.-Ing.) and an M.Sc. in computer studies (Dipl.-Inform.) from Technical University in Berlin, Germany. He is also a visiting lecturer at Humboldt University in Berlin.

His publication record includes more than 60 papers on automotive software engineering, Model-Based Design and safety-related software. He is a member of the Special Interest Group for Automotive Software Engineering in the German Computer Society (GI-ASE) and a former member of the ISO 26262 subworking group on software.

E-mail: Mirko.Conrad @ mathworks.com

Guido Sandmann
Automotive Marketing Manager, EMEA
The MathWorks GmbH
Adalperostr 45
85737 Ismaning

E-mail: Guido.Sandmann@mathworks.de