

Using Simulink® and Real-Time Workshop® Embedded Coder for IEC 61508 Applications

Mirko Conrad, Dr.-Ing

This paper is an updated version of a paper that originally appeared in the proceedings of the Workshop on Model-Based Development of Embedded Systems III (MBEES 2007), Schloß Dagstuhl, Germany, Jan. 2007.

August, 2007

ABSTRACT

When developing safety-critical embedded software it is important to consider the objectives of standards such as IEC 61508. These standards impose additional constraints on the development processes and require the production of evidence that the objectives were met.

IEC 61508 is a generic safety publication. The standard was developed for traditional (hand-coded) software development processes and does not cover advanced software development technologies such as Model-Based Design and production code generation. Therefore the measures and techniques recommended by IEC 61508-3 need to be mapped onto Model-Based Design processes and tools.

This paper discusses the usage of Model-based Design tools from The MathWorks for safety-critical embedded applications and outlines a solution to facilitate the demonstration of compliance with the IEC 61508 objectives.

KEYWORDS

IEC 61508, Model-Based Design, production code generation, software safety integrity tables, Simulink, Real-Time Workshop Embedded Coder

1 MODEL-BASED DESIGN OF EMBEDDED SOFTWARE AND IEC 61508

Model-Based Design is becoming the preferred software engineering paradigm for the development of embedded software in various industries. The main advantages of this approach are that software development time is reduced and product innovation is enhanced through the use of executable specifications and code generation. These technologies are successfully used to produce software for safety-critical applications, see e.g. [Pot04, TMW05], but extra consideration is needed to address the constraints imposed by standards and to produce the required evidence for compliance demonstration.

IEC 61508 'Functional safety of electrical / electronic / programmable electronic safety-related systems' [IEC 61508] is a sector-independent safety publication for electrical / electronic / programmable electronic safety-related systems (short: E/E/PES). The standard was developed originally by the process and automation industries. Its seven parts (referenced as IEC 61508-1 to IEC 61508-7) were published between 1998 and 2000, i.e. in the early Model-Based Design era. IEC 61508-3 focuses on software. Software failures are viewed as a result of faults systematically

introduced during software development. In recognition of this, IEC 61508-3 defines requirements and constraints for the software development, verification, validation and testing processes [MK06].

IEC 61508 can be considered as a prescriptive standard, which provides detailed lists of techniques and measures with recommendations. The degree of rigor required in software development, verification, validation and testing depends on the criticality of the software within the embedded application and is expressed in terms of safety integrity levels (SILs). To aid the selection of software safety techniques and measures appropriate to the required safety integrity IEC 61508 provides so called software safety integrity tables ranking the various techniques/measures against the four safety integrity levels SIL 1 to SIL 4 (see annex A of IEC 61508-3).

As an example IEC 61508-3 highly recommends the usage of language subsets for SIL 3 and 4 applications (cf. IEC 61508-3, Table A.3). The related extract of the safety integrity table A.3 is given in Table 1.

Table 1: Example of a software safety integrity table as provided by IEC 61508-3 (extract)
A.3: Software design and development: Support tools and programming language

Technique / Measure	SIL1	SIL2	SIL3	SIL4
...
3 Language subset	o	o	++	++
...

- ++ ... The technique or measure is highly recommended for this SIL.
- o ... The technique or measure has no recommendation for or against being used.

Altogether, annexes A and B of IEC 61508-3 include 19 software safety integrity tables recommending 120+ measures and techniques (see the appendix of this paper for an overview). Annex C of IEC 61508-7 provides detailed descriptions of selected measures and techniques.

The software safety integrity tables are intended to be tailored for a particular safety critical application, i.e. it has to be documented which measures/techniques were (partially) implemented and / or which replacement measures were employed. As a rule, if a particular highly recommended technique or measure is not used then the rationale behind not using it should be documented and agreed with the safety assessor. The tailoring process for the software safety integrity tables is illustrated by means of worked examples in annex E of IEC 61508-6. The tailored software safety integrity tables are used within the compliance demonstration process and are necessary for system certification. They are used as evidence that the objectives of the standard were met.

The tailoring is a lengthy and labour intensive process. However, the tailored tables for similar projects typically share common parts and according to [CD06, CD06a] the project-specific accruing effort for the tailoring can be reduced effectively by factoring out these common parts. This can be realized if adapted tables covering typical development projects and typical tool chains are made available.

Since IEC 61508 was written with traditional (hand-coded) software development processes in mind, the standard does not cover advanced embedded software development technologies such as Model-Based Design and production code generation. Therefore, the recommended techniques and measures are targeted at hand-coded development procedures and need to be mapped onto Model-Based Design processes and tools making the tailoring more complex. For example, the recommendation for language subsets in Table A.3 leaves open if this is related to the modeling language, e.g. Simulink®, or the implementation language, e.g. C, or both.

2 COMPLIANCE DEMONSTRATION

In order to ensure a consistent interpretation of the standard across various Model-Based Design projects and to avoid unnecessary repeated work in projects comparable to each other, adapted tables covering typical Model-Based Design workflows and industry standard tools such as the Simulink [SL] and Real-Time Workshop® Embedded Coder [RTW-EC] can be used.

The project-specific accruing effort for this can be reduced effectively if respective generic versions (a.k.a. templates) of these tables for typical Model-Based Design projects are made available. These templates are then instantiated project-specifically.

In order to create the template tables, the original software safety integrity tables from the standard are extended by an additional column labeled 'Applicable Model-Based Design Tools and Processes'. The entries in the column describe how a particular measure or technique can be supported by products or solutions for Model-Based Design. Typically the applicable Tools / Processes for Model-Based Design are split up into model level and code level activities.

Table 2 shows an extract of an annotated IEC61508-3 table which corresponds to the example in Table 1. The rationale behind the entries of the additional column is as follows: In Model-Based Design language subsets occur on the design level (a.k.a., modeling language subset) as well as on the code level (a.k.a., implementation language subset). Pure subsetting of the modeling and / or the implementation language is inadequate in many cases. Furthermore, compliance to an implementation language subset shouldn't be enforced only after the code has been generated. Instead it is important to have modeling and code generation guidelines that restrict the use of blocks and suggest proper code generation settings. Making recommended patterns and style guides available has also proven useful.

Table 2: Example of a generic, annotated software safety integrity table for Model-Based Design projects (extract)
A.3: Software design and development: Support tools and programming language

Technique / Measure	SIL 1	SIL2	SIL3	SIL4	Applicable Model-Based Design Tools and Processes
...
3 Language subset	o	o	++	++	<p>The MAAB Style Guides and/or organization specific modeling guidelines can be used to define a subset of the modeling language.</p> <p>The Simulink Block Data Type Support section of the Simulink documentation lists the blocks, which can be used for code generation with Real-Time Workshop Embedded Coder.</p> <p>MISRA-C and/or organization specific coding guidelines can be used to define a subset of the implementation language.</p> <p>Restricted language subsets can be partially enforced by using Simulink Model Advisor</p> <p>Restricted modeling language subsets can be enforced by model reviews based on reports generated by Simulink Report Generator.</p> <p>PolySpace for C/C++ can be used to verify MISRA-C: 2004 compliance.</p> <p>PolySpace for C/C++ can be used to check language subset considerations within the generated code.</p> <p>Restricted implementation language subsets can be enforced by code reviews based on Real-Time Workshop Embedded Coder – Code Generation Reports.</p>
...

According to [Bär05], MISRA-C [MISRA04] compliance is an implementation of a coding standard as required by IEC 61508-3. TÜV Süd accepts MISRA-C, if adherence to the MISRA-C rules can be shown and the compliance to the coding guidelines will be verified by a static code analysis tool. The OEM Initiative Software (HIS) also recommends to use the entire set of MISRA-C:2004 guidelines. For exceptional cases deviations are to be documented following the deviation procedure described in section 4.3.2 of the actual MISRA-C document [HIS06].

To address MISRA-C compliance, MathWorks maintains a MISRA-C compliance analysis package for Real-Time Workshop Embedded Coder since Release 12.1. This package is based on model inspections, code analysis, and quality engineering product tests suites. The MISRA-C compliance package includes an overview document, a compliance matrix, and presentation of violations and mitigations. Simulink and Stateflow model examples as well as Real-Time Workshop Embedded Coder code examples are also included.

Simulink is a general purpose visual modeling language. It can be used to create controller as well as plant models. So not all modeling features and settings are appropriate for modeling embedded software. The MAAB style guidelines [MAAB01] were developed to address this issue. They became a popular source of information for the derivation of company specific modeling language subsets, pattern and guidelines. For safety-critical applications additional patterns and guidelines might be useful.

In summary, the MISRA-C and MAAB guidelines are means to partially fulfill the IEC 61508-3 requirements on language subsets and coding standards. To verify guidelines compliance, powerful verification and validation capabilities are essential.

Based on the generic template tables for Model-Based Design such as illustrated in Table 2 the project specific effort can be reduced to the documentation of the deviations from the generic version. In particular it is necessary to document which measures were implemented and/or which replacement measures were employed. The following steps apply:

1. Selection of the SIL and removal of the columns for the non-applicable safety integrity levels.
2. Selection of a set of measures / techniques for the particular application (in case of alternate or equivalent techniques/measures).
3. Renaming of the column 'Applicable Model-Based Design Tools and Processes' into 'Interpretation in this application'.
4. Labeling of the measures / techniques for the particular application depending on whether they are 'used', 'used to a limited degree' or 'not used'. If a particular measure / technique is not used, a reason has to be provided and the Applicable Tools / Processes for Model-Based Design belonging to it need to be removed.
5. Application-specific modification/refinement of the remaining annotations in the column 'Interpretation in this application'

Table 3 exemplifies the results of the customization process for the language subset part of Table 2.

The derived application-specific tables are supposed to be submitted to the certification authority as part of the compliance demonstration process. It is recommendable to submit a first version of the tables early in the life cycle in order to be discussed and agreed with the assessor.

Note, that compliance with the IEC 61508 standard does not ensure the safety of the software or the application. But it can be demonstrated that state-of-the-art procedures were applied.

Table 3: Example of a derived application-specific table (extract)
A.3: Software design and development: Support tools and programming language

Technique / Measure	SIL3	Interpretation in this application
...
3 Language subset	++	Used: The MAAB Style Guides are used as subset of the modeling language. MISRA-C:2004 is used as subset of the implementation language. The modeling language subset is as far as possible enforced by using Simulink Model Advisor . All reported deviations and modeling language considerations which cannot be checked automatically need to be manually reviewed. PolySpace for C/C++ is used to verify MISRA-C: 2004 compliance. All reported deviations need to be manually reviewed.
...

3 SUMMARY AND CONCLUSION

Embedded software is increasingly used for safety-related or safety-critical applications. Accordingly, engineers require embedded software development tools and processes that include proven, state-of-the-art quality assurance measures. A means to meet this requirement is to base the embedded software development process upon the IEC 61508 safety standard.

IEC 61508-3 requires systematic software engineering processes and additional verification, validation and testing measures in order to facilitate product safety and to minimize the remaining risk.

Model-Based Design with production code generation can be used with IEC 61508. An appropriate combination of different products from the Simulink product family addresses a broad spectrum of the IEC 61508-3 objectives related to software. MathWorks' products including Simulink, Stateflow, and Real-Time Workshop Embedded Coder have been successfully deployed for safety-critical applications up to the highest criticality levels.

However, since IEC 61508 dates back to the early Model-Based Design area and does not address popular development technologies such as production code generation directly, it has been subject to interpretation. Mapping onto development practices using Model-Based Design can be a time-consuming process.

In the past, the mapping of the IEC 61508 objectives onto Model-Based Design processes and tools was done on a per project basis. Significant portions of this mapping were caused by bridging the gap between the standard and modern software development approaches and not by the specifics of the project under consideration. This resulted in unnecessary high efforts for the individual projects.

This paper proposed a more standardized way to facilitate and document IEC 61508 conformant Model-Based Design processes. First, the software safety integrity tables of IEC 61508 were enhanced in a way that they provide a project independent mapping of recommended techniques / measures onto tools of the Simulink product family and related processes. These template tables can then be easily tailored according to the needs of a particular project. The application-specific tables resulting from the tailoring can be used as evidence within the compliance demonstration process as suggested by IEC 61508-6.

Providing annotated versions of the software safety integrity tables covering typical Model-Based Design workflows is a powerful means to offer guidance for Model-Based Design projects that must meet the IEC 61508 objectives.

In addition, using this approach cuts down the project specific efforts for compliance demonstration and reduces time-to-system-certification.

The outlined approach is not restricted to IEC 61508, is also applicable to derived sector specific standards such as the upcoming ISO 26262 [ISO 26262, FP06].

Please contact the author for further discussion.

3 REFERENCES

- [Bär05] Andreas Bärwald: IEC 61508 & MISRA C - The Benefits of Utilising IEC 61508 and MISRA C for Automotive Applications. 1st IEE Automotive Electronics Conference, London, UK, 2005
- [CD06] Mirko Conrad, Heiko Dörr: Deployment of Model-based Software Development in Safety-related Applications - Challenges and Solutions Scenarios. Proc. Modellierung 2006, Innsbruck, Austria, 2006, LNI Vol P-82, pp. 245-254
- [CD06a] Mirko Conrad, Heiko Dörr: Einsatz von Modell-basierten Entwicklungstechniken in sicherheitsrelevanten Anwendungen: Herausforderungen und Lösungsansätze. Proc. Dagstuhl-Workshop 06022 Modellbasierte Entwicklung eingebetteter Systeme II (MBEES'06), Schloß Dagstuhl, Germany, 2006, pp. 1-17
- [FP06] Matthias Findeis, Ilona Pabst: Functional Safety in the Automotive Industry, Process and methods. VDA Winter meeting, 2006
- [HIS06] HIS Working Group Software Test: Gemeinsames Subset der MISRA C Guidelines. Version 2.0, 2006
- [IEC 61508] IEC 61508-3:1998. International Standard IEC 61508 Functional safety of electrical/electronic/ programmable electronic safety-related systems. 1st edition, 1998
- [ISO 26262] ISO 26262:2005. Road vehicles - Functional safety: Working Draft, 2005
- [MAAB01] MathWorks Automotive Advisory Board: Controller Style Guidelines for Production Intent Development Using MATLAB, Simulink, and Stateflow. Version 1.00, 2001
- [MISRA04] MISRA-C:2004. The Motor Industry Software Reliability Association: Guidelines for the use of the C language in critical systems. 2004
- [MK06] J. McDermid and T. Kelly. Software in safety critical systems: Achievement and prediction. Nuclear Future, 02(03):140–145, 2006.
- [Pot04] Bill Potter: Use of The MathWorks Tool Suite to Develop DO-178B Certified Code. ERAU / FAA Software Tools Forum, Daytona Beach, FL., US, 2004
- [RTW-EC] Real-Time Workshop® Embedded Coder User's Guide. Version 4, The MathWorks Inc, 2006
- [SL] Using Simulink®. Version 6, The MathWorks Inc, 2006
- [TMW05] Alstom Generates Production Code for Safety-Critical Power Converter Control Systems, User Story, The MathWorks, 2005
www.mathworks.com/products/rtwembedded/userstories.html?file=10591

Appendix: Overview of IEC 61508-3 Software Safety Integrity Tables

IEC 61508-3 Annex A Tables

IEC 61508-3 Annex B Tables

A.1: Software safety requirements specification

- 1 Computer-aided specification tools
- 2a Semi-formal methods
- 2b Formal methods

A.2: Software design and development: software architecture design

- 1 Fault detection and diagnosis
- 2 Error detecting and correcting codes
- 3a Failure assertion programming
- 3b Safety bag techniques
- 3c Diverse programming
- 3d Recovery block
- 3e Backward recovery
- 3f Forward recovery
- 3g Re-try fault recovery mechanisms
- 3h Memorising executed cases
- 4 Graceful degradation
- 5 Artificial intelligence - fault correction
- 6 Dynamic reconfiguration
- 7a Structured methods
- 7b Semi-formal methods
- 7c Formal methods
- 8 Computer-aided specification tools

A.3: Software design and development: Support tools and programming language

- 1 Suitable programming language
- 2 Strongly typed programming language
- 3 Language subset
- 4a Certified tools
- 4b Tools: increased confidence from use
- 5a Certified translator
- 5b Translator: increased confidence from use
- 6 Library of trusted / verified software modules and components

A.4: Software design and development: detailed design

- 1a Structured methods
- 1b Semi-formal methods
- 1c Formal methods
- 2 Computer-aided design tools
- 3 Defensive programming
- 4 Modular approach
- 5 Design and coding standards
- 6 Structured programming
- 7 Use of trusted/verified software modules and components

B.7: Semi-formal methods

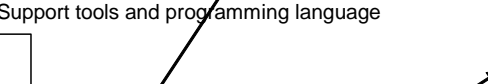
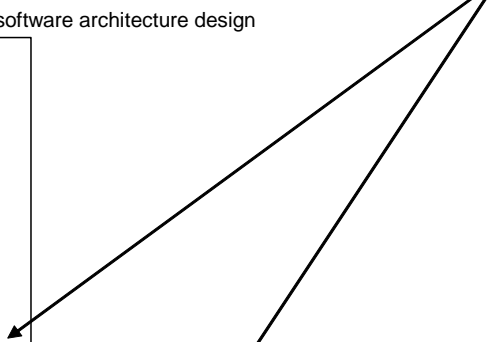
- 1 Logic/function block diagrams
- 2 Sequence diagrams
- 3 Data flow diagrams
- 4 Finite state machine/ transition diagrams
- 5 Time Petri nets
- 6 Decision/truth tables

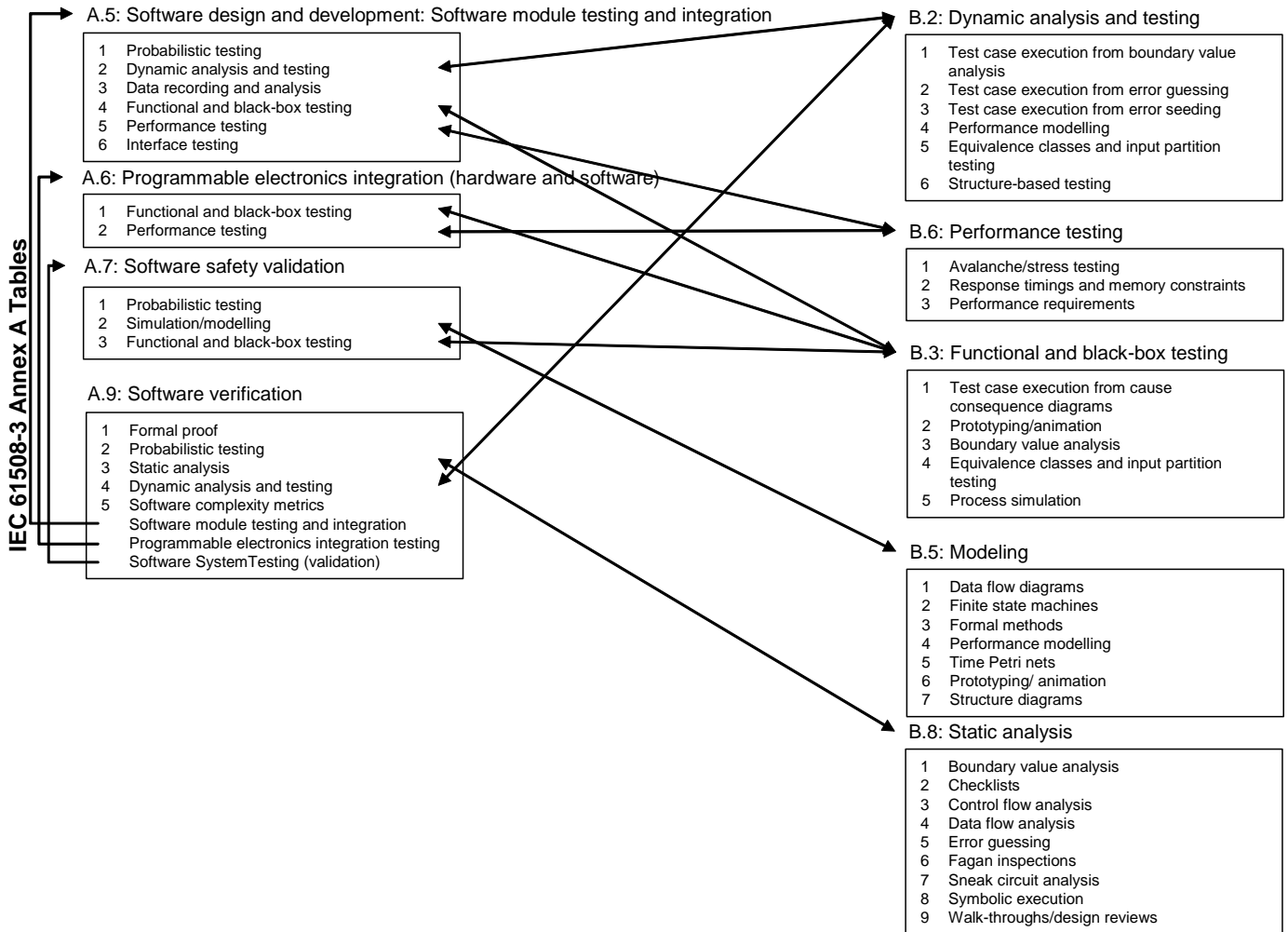
B.9: Modular approach

- 1 Software module size limit
- 2 Information hiding/ encapsulation
- 3 Parameter number limit
- 4 One entry/one exit point in subroutines and functions
- 5 Fully defined interface

B.1: Design and coding standards

- 1 Use of coding standards
- 2 No dynamic objects
- 3a No dynamic variables
- 3b Online checking of the installation of dynamic variables
- 4 Limited use of interrupts
- 5 Limited use of pointers
- 6 Limited use of recursion
- 7 No unconditional jumps in programs in higher level languages





A.8: Modification

- 1 Impact analysis
- 2 Reverify changed software module
- 3 Reverify affected software modules
- 4 Revalidate complete system
- 5 Software configuration management
- 6 Data recording and analysis

A.10: Functional safety assessment

- 1 Checklists
- 2 Decision/truth tables
- 3 Software complexity metrics
- 4 Failure analysis
- 5 Common cause failure analysis of diverse software (if diverse software is actually used)
- 6 Reliability block diagram

B.4: Failure analysis

- 1 Test case execution from boundary value analysis
- 2 Test case execution from error guessing
- 3 Test case execution from error seeding
- 4 Performance modelling
- 5 Equivalence classes and input partition testing
- 6 Structure-based testing

