

Applying Model-Based Design and Automatic Production Code Generation to Safety-Critical System Development

Dirk Fleischer, Michael Beine and Ulrich Eismann
dSPACE GmbH

Copyright © 2009 SAE International

ABSTRACT

Model-based software development and automatic code generation have become increasingly established in recent years. The automotive industry has widely adopted and successfully deployed these methods in many different series production programs worldwide. This brought various benefits, such as a reduction in development times, improved quality due to more precise specifications, and early verification and validation by means of simulation.

At the same time, more and more safety-related and safety-critical systems have been - and will be - introduced into modern vehicles. Common examples are active front steering, adaptive cruise-control, and integrated chassis control.

This leads to the question, if and how model-based design and automatic production code generation can be applied to the development of safety-critical systems.

INTRODUCTION

FUNCTIONAL SAFETY - More and more safety-related or safety-critical functions are being introduced into new vehicles, mainly found in the domains of active safety systems, driver assistance systems and vehicle dynamics, e.g. active steering or braking systems. Functional safety is a major requirement for these functions.

Functional safety in general means not exposing a user to intolerably dangerous system states. This of course cannot be achieved solely by software. Appropriate system and hardware concepts have to be applied, including redundant or even diversified implementation for input signals to the function as well as mechanical or electronic means to deal with hazards that occur, etc. This paper focuses on the software part of functional safety, particularly on developing automotive software using model-based design approaches incorporating automatic code generation.

SAFETY STANDARDS – Standards that apply to automotive software development are IEC 61508 [1] and particularly the upcoming ISO 26262 [2]. IEC 61508 is a generic across-the-industries standard that encourages the derivation of industry-specific standards. It originated in the process control automation industry, and sector-specific standards have already been derived for the process industry (IEC 61511), nuclear power plants (IEC 61513) and machinery (IEC 61513). Upcoming ISO 26262, which reached ISO Committee Draft (CD) status in 2008, will be a derivative that is especially tailored to the automotive industry. Where IEC 61508 has to remain abstract in many points, ISO 26262 can be far more specific with regard to automotive electronics development. IEC 61508 was created and published in the late 1990s, before model-based design and code generation became widely adopted. It can therefore give little direct advice of how to comply within a model-based development process. The standard has therefore to be interpreted. ISO 26262 will also address this issue and cover specifically model-based design aspects, reflecting the importance of this approach in automotive software development today.

SAFETY INTEGRITY LEVELS – Both standards provide classification schemes for ranking functions according to the risk of the most hazardous event the user can be exposed to by the system. In IEC 61508, safety integrity levels are numbered from SIL1 to SIL4 with increasing risk, meaning SIL4 is the highest safety integrity level. In ISO 26262 safety integrity levels range from ASIL A to D, also with increasing risk. Requirements for the development process, verification and validation as well as tool selection depend on the safety integrity level a function to be developed was classified for.

MODEL BASED DESIGN – The automotive industry has widely adopted model-based design for several reasons. One of them is the concept of executable specifications [3] in the form of simulatable models which allow early verification and validation. Another major reason is that automatic code generation can close the gap between function design and software implementation. It not only streamlines the development process but makes the transition from an executable specification to actual software implementation more consistent.

PRODUCTION CODE GENERATION – There are already a number of safety-critical applications that have been developed using the dSPACE TargetLink® code generator for production code generation. For example, early use of TargetLink was made as far back as 2002, for a cabin-pressure control system in the aerospace industry [4] and for IEC 61508 SIL3 level classified automotive systems [5]. An example of a more recent use is BMW [6,7] which developed safety-critical vehicle dynamics systems. The question of how to comply with safety-standard requirements was answered differently

in these projects. In the first two cases, complete verification and validation of the code generator's output was performed and in the first case the code was certified according to RTCA DO-178B [8], while in the second case it was certified according to IEC 61508 SIL3. In the last case BMW qualified TargetLink using a validation suite approach [9],

This paper will show how model-based design can be applied to the development of safety-critical systems and what should be considered for the selection of a code generator.

MODEL-BASED DEVELOPMENT PROCESS

Model-based development today is mostly associated with the function level, dealing with single functions of a vehicle or ECU. However, the model-based design paradigm is increasingly being extended to the system level of whole ECUs and ECU networks, where software architectures and systems are modeled. In this paper we will focus on the model-based development process on the function level.

Model-based development incorporating automatic code-generation has to cover two levels, the model level for function design which serves as an input to the code generation process, and the code level as an output in the form of function implementation, usually as C code. Within this paper we will not only distinguish between model and code level, but also between modeling environment and code generator. As concrete examples we will use the most widespread combination of modeling environment and code generator in the automotive industry: MATLAB®/Simulink®/Stateflow® and dSPACE TargetLink. Third-party tools complementing the development process will also be mentioned where required.

When developing safety-critical software, compared to non-safety-related software development, additional requirements specified by relevant safety-standards have to be met. Meeting these requirements should also be considered to be best practices for non-safety-related software.

IEC 61508-3 recommends various techniques and measures for the safety-critical software development process. These are detailed in 10 top-level tables (Tables A.x) and 9 detailed tables (Tables B.x), which expand some of the topics of the top-level tables. Not all measures directly apply to a model-based design approach or are directly supported by it - which does not mean they could not be realized by it. ISO 26262 (Part 6 in current committee draft version) follows a comparable approach by providing tables of methods and measures contributing to the fulfillment of requirements for the software level of product development.

Table A.5 – Software design and development:
software module testing and integration (see 7.4.7 and 7.4.8)

Technique/Measure*	Ref	SIL1	SIL2	SIL3	SIL4
1 Probabilistic testing	C.5.1	---	R	R	HR
2 Dynamic analysis and testing	B.6.5 Table B.2	R	HR	HR	HR
3 Data recording and analysis	C.5.2	HR	HR	HR	HR
4 Functional and black box testing	B.5.1 B.5.2 Table B.3	HR	HR	HR	HR
5 Performance testing	C.5.20 Table B.6	R	R	HR	HR
6 Interface testing	C.5.3	R	R	HR	HR
NOTE 1 – Software module and integration testing are verification activities (see table A.9).					
NOTE 2 – Appropriate techniques/measures shall be selected according to the safety integrity level.					
* A numbered technique/measure shall be selected according to the safety integrity level.					

Figure 1: Example table from IEC 61508 part 3 (HR = highly recommended, R = recommended, --- = no recommendation)

REQUIREMENTS TRACEABILITY

IEC 61508-3 (Table A.1 #1) defines computer-aided specification tools as one measure to be applied to the development process. Requirements management today is not covered specifically by model-based design tools. Closely linked to requirements management is the traceability of requirements throughout the development process. Not least to facilitate requirements based testing, as required by ISO 26262 for all ASILs. Following the distinction of modeling environment and code generator, a major part of requirements traceability ideally lies in the modeling environment providing the initial bidirectional, navigable links from external requirements management tools to the model, and providing additional consistency checks for the linked requirements as well as appropriate report generation. In order to achieve full traceability the code generator must establish the links between the model as input and the code as output. This can be achieved, for example, by generating C code in HTML format, including requirement-specific hyperlinks to the model and/or the original location in the requirements management system. Simulink combined with the Verification and Validation Toolbox [10] as a modeling environment supports requirements traceability from external requirements management tools to the model level. TargetLink as a code generator provides links between model and code that also support tracing requirements.

SOFTWARE ARCHITECTURE

Architecture itself is addressed only to a limited extent by IEC 61508 but is now a crucial point in automotive software development due to the highly increased complexity of software, and therefore will be addressed by ISO 26262 in greater detail. However, IEC 61508-3 does address software modularization, for instance (Tables A.4 #4 and B.9) as one concept of software architectures.

MODULARIZATION – Simulink as a modeling environment itself supports the hierarchical and modular partitioning of models. The Simulink Verification and

Validation Toolbox further supports the measurement of model complexity on system and subsystem level. It therefore helps to achieve suitable module sizes. The code generator on the other hand must provide features that allow modularization on code level as well. TargetLink gives the user various means to configure whether model parts should be realized as separate functions, separate C-code files, etc. Incremental code-generation additionally allows even smaller parts of the model to be coded separately, with the benefit that changes in other parts of the model do not affect the already incrementally generated code which then does not need to be verified again.

MODELING AND CODING GUIDELINES

In general, but particularly for safety-critical applications programming standards describing good programming style and avoiding unsafe language features should be used. IEC 61508-3 (Table A.3 #3) as well as ISO 26262 highly recommend using a safe language subset for the higher safety integrity levels. MISRA C [11] is such an established standard, defining a safe language subset for the C language. But which guidelines address the model level?

MAAB GUIDELINES – The MathWorks Automotive Advisory Board (MAAB) guidelines [12] are a collection of rules following several objectives such as increasing readability, smoothening workflows, and enabling design for verification and validation as well as for code generation. In a first step they also limit the set of blocks provided by the modeling environment MATLAB/Simulink to a set of blocks to be used for controller design. This is similar to the approach followed by the TargetLink code generator, providing a dedicated library for controller development. Still, in order to really address the requirements of safety standards, additional guidelines have to be used.

TARGETLINK GUIDELINES – The TargetLink Modeling Guidelines [13] are a collection of rules pursuing objectives such as efficient workflow, readability, and efficient code generation, but also increased reliability and safety of models, e.g. by complying with MISRA C.

MISRA AC TL GUIDELINES – At the end of 2007 MISRA published the MISRA AC TL guidelines [14], which are specifically designed for the TargetLink code generator. The primary motivation of the MISRA AC TL guidelines is helping to develop safety-critical systems. MISRA AC TL provides guidance for using TargetLink to help users avoid pitfalls on both model and code level. MISRA AC TL guidelines can be seen as the TargetLink counterpart to MISRA-C:2004 guidelines for manually created C code. TargetLink was the first – and so far only – code generator such guidelines have been published for by MISRA.