Tools and Methods for Validation and Verification as requested by ISO26262

Markus Gebhardt, Axel Kaske – ETAS GmbH
Markus.Gebhardt@etas.com
Axel.Kaske@etas.com

1 Abstract

The following article will have a look on methods for validation and verification of software requested for safety related systems by ISO26262 (1) (or similar standards) and will point out how some dedicated tools from ETAS may help to fulfill and implement these. A brief introduction into the underlying technology will be given in order to discuss the aspects/use cases where these tools can be used either for simulation purpose or in combination with the final target.

2 Introduction to ISO26262

ISO26262 is the automotive derivative or variant of the IEC61508 (an international standard) relating to the functional safety of electrical/electronic/programmable electronic (E/E/PE) safety-related systems. In this context, a system is defined to include sensors and other input devices, the programmable electronics itself and all actuators and other output devices (see (2) for further details). ISO26262 will be released in July 2011. This obliges that all products (vehicle up to 3,5t) have to be developed/produced compliant to ISO26262 from that point in time. Otherwise this might lead to difficulties in case of warranty claims.

The standard consists of 10 parts, covering the full lifecycle of E/E/PE safety related systems from functional safety management over concept, design and development to production and operation. This article will focus on part 6, product development at the software level with a special focus on software for electronic control units (ECU).

2.1 ISO26262 with respect to ECU Software

ISO26262 requires that all tools used in the development process for software of the safety critical system have to be classified in order to assess the impact of errors that could be introduced by a malfunction of a tool (tool impact, TI). Depending on the tool impact level, a required tool confidence level (TCL) must be derived depending on the probability whether an error introduced can be detected within the development process (tool error detection, TD). For higher tool confidence levels (that is, critical errors can be introduced by the tool and they are hard to detect) a qualification process for tools may become necessary.

An automotive safety integrity level (ASIL) is assigned to the E/E/EP system to be developed is derived depending on severity, controllability and exposure of a malfunction of the system. All derived measures in the development process depend on this ASIL of the system. The higher the ASIL of the system under development, the higher the required confidence level for the tools used in the development process is.

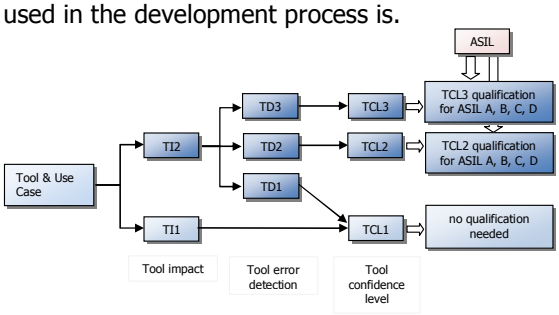

Figure 1: Tool classification and qualification process

By increasing the tool error detection – that means increasing the level of detecting errors in the software development– a lower TCL can be reached. (Confusingly, a higher confidence in detecting erroneous output of the tool is expressed in lower TDs). This can help to avoid the need of a tool qualification or at least to lower the level of tool qualification needed for a defined ASIL of the product under development, therefore reducing effort and costs.

In addition to this, mechanisms for error detection on all levels of the software development in general are requested or strongly recommended by the ISO 26262.

In part 6 of the ISO26262 standard, these necessary mechanisms are listed for software architectural design [part 6/chapter 7], software unit design and implementation

[part 6/chapter 8], software unit test [part 6/chapter 9] and software integration and testing [part 6/chapter 10]. These methods shall be applied to achieve the necessary level of safety and build up confidence in the tested artifacts.

2.2 Validation and verification tools

While some of the methods suggested require manual work, the majority can benefit or are even only possible by the use of verification and validation tools supporting these methods. In the following we will explain how tools that do execution of C-code instead of simulation can serve this purpose, especially when several use cases can by covered with the same C-code.

Obviously, a tool used to assess the quality of an artifact must not malfunction, too, or errors might pass undetected and a false evidence of safety is created, which might be even worse than an erroneous output. Therefore, tools used for validation and verification must undergo the same classification and qualification process as the tools used to create and develop the software and configuration or calibration data explained in the last chapter. Consequently, ISO26262 requires at least the same level of confidence in these validation and verification tools as for the tool generating the artifacts to be tested.

3 Validation and verification with prototyping tools

3.1 Virtual prototyping

Modern development processes for ECU software usually use model based development tools to specify the control algorithms. Simulation of functional modules or models is normally done directly in the behavior modeling tool respectively in its associated simulation environment. This is so far no problem, because all necessary measurements, even inside the model, are possible and tests/metrics like model-coverage (MC) or decision-coverage (DC) can be done easily by tracing the model (see (3)).

But it becomes a problem if instead of the functional model the "real" C-code of the software component (SWC) shall be simulated. In this case the above mentioned simulation environments integrate in general the SWC simply as a black box and measurements are only possible at the interface level of the SWC (in-/out ports). This is enough to compare the general behavior in the good case. But already debugging is difficult because it is almost impossible to understand the signal flow inside the SWC as there are no internal measurement points available. It becomes even more difficult if you want to measure DC as you need to trace the C-code. And as shown in (4) & (5) functional equivalent C-code can lead to different test case, which means, even if the SWC behaves as expected you might need created additional test cases depending on your implementation in order to get the same coverage level.

Another restriction of the standard simulation environment of behavior modeling tools is that they do not provide an ECU like integration environment for the SWCs with access to basic software modules like ECU diagnostics managers, mode managers, network managers etc. (see AUTOSAR SW architecture (6)). But this is in general necessary as almost every SWC call at least one of those basic SW modules. Therefore you have to provide the OS and other basic software interface by further simulation elements which might not be sufficiently representative.

These restrictions become especially crucial for AUTOSAR developments.

To overcome several of these restrictions we want to point out how the concept of tool INTECRIO can help you.

INTECRIO is an integration and build tool which uses for all supported targets an OSEK OS or an AUTOSAR-OS in combination with an AUTOSAR-RTE.

All integration is done on C-code level. To do so, the functional models are converted by the code generators of the behavior modeling tools into non-optimized, floating point code while SWC use (normally) highly optimized fix point code

For integration INTECRIO needs always a file triple

1. C-code (*c. and *.h files)
2. A2L file as description file for all measurable variables of the model (equal C-code) and all calibratable parameters/curves/maps of the model (equal C-code)
3. Interface description file which can be of course AUTOSAR (SWC description) or an ETAS proprietary XML format (SCOOP-IX (7)) for all other model/modules

For the standard behavior modeling tools like Simulink or ASCET this file triple is generated automatically, for any other C-code (hand-written or machine generated) a tool called INCODIO from Systecs can help to generate the file triple.

As INTECRIO automatically provides the automotive OS etc. especially SWCs can be integrate without adaptation and do run in an ECU like environment.

As INTECRIO integrates on C-code level you are able to use C-code debuggers/development environments to trace the code etc. in order to

measure metrics like code/decision coverage etc. easily. This helps to understand if the conversion of function model to C-code requires additional test cases or not.

Therefore INTECRIO can be used to fulfill the methods proposed by ISO26262 as shown below in 7 Appendix.

3.2 Rapid Prototyping

However, the algorithm under development must be not only functionally correct, but must also work correctly under real time conditions. Real time means the algorithm must be computed with the same timing constraints that apply for the control unit executing the algorithm in the final product. Inputs and outputs must be acquired and served under exactly the same conditions. As target environments to be used for both software unit and software integration testing, ISO26262 lists model-in-the-loop tests, software-in-the-loop tests, processor-in-the-loop tests and hardware-in-the-loop tests [part 6, sections 9.4.6 and 10.4.8]. For unit testing of automatically generated code, back-to-back tests between the model and the generated and compiled object code are suggested.

Rapid Prototyping exists mainly as two use cases. One is fullpass where all functions/SW is executed on rapid prototyping HW and additionally all bus communication and physical I/O is also done via rapid prototyping HW.

Figure 2: Rapid Prototyping Fullpass system running all functions/SWC and controlling all I/O

Second use case is bypass, where in general only a part of the functions/SW (e.g. the new function/SW) is executed on the rapid prototyping HW while the other part (e.g. the "old", existing part) of function/SW is running on the ECU. The same applies for the bus communication and physical I/Os where only the new part is taken over by the rapid prototyping system in order to ease the setup of the entire system (see figure 4).

In Rapid Prototyping, bypass techniques are widely used as an efficient way to improve algorithms of an ECU by bypassing the ECU calculation with an executable specification on rapid prototyping hardware. In model based development, an executable specification is available and used for this method. If the bypass interface implementation in the ECU software allows capturing the results of the ECU function in parallel to executing the bypass, this can also be used as a back-to-back test, if the code of the bypassed ECU function has been created from the model itself.

In order to integrate additional functions (meaning C-code) into an existing ECU, ETAS offers a tool called EHOOKS which provides a sophisticated configuration, build and patching mechanism.

During configuration one is able to define which ECU variables shall be read into the new ECU process or SWC and which ECU variables shall be overwritten by the new one. Additionally one can specify the scheduling of the process (e.g. runnables being called each).

First EHOOKS looks for all write accesses to the selected variables in the ECU code, then EHOOKS patches the original HEX file to integrate the automatically compiled C-code. Additionally the A2L file is enhanced with control variables and optionally additional variables for the result of the original ECU calculation. With comparing the results of the executable specification calculated on the external rapid prototyping hardware and the copied result of the ECU's computation of the implemented code, back to back tests using the target hardware and execution environment can be performed.

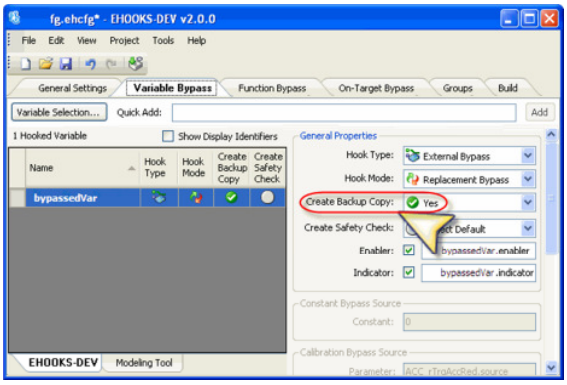

Figure 3: Using EHOOKs as tool for instrumenting the compiled ECU software for external bypass to perform back-to-back testing

The big advantage of EHOOKS is that you do not need the original C-code of the basic ECU. New SWCs can be integrated in the ECU code within a few minutes and this integrated SWC behaves identically to a classic integration via complete build (one jump/return instruction is needed additionally).

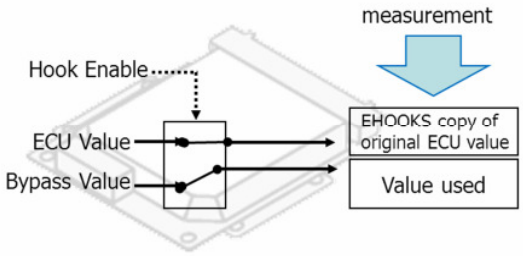

Figure 4: EHOOKS provides variable copies for back-to-back test.

Also for these rapid prototyping approaches INTECRIO is ideally suited as it can integrate the same models/C-code as for virtual prototyping with real I/O hardware still using an OSEK-OS etc. to provide an ECU like environment (especially necessary for AUTOSAR SWCs).

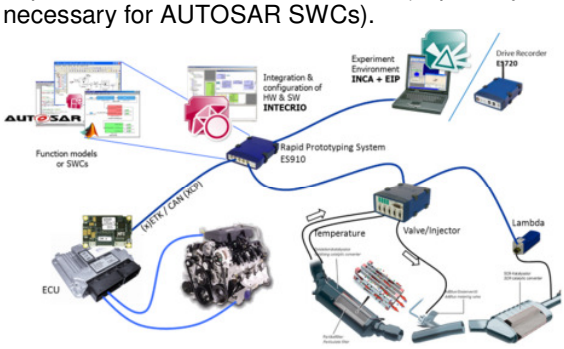

Figure 5: Rapid Prototyping Bypass system running only a few (new) functions/SWC and controlling only the I/O of the new sub-system (filter and injector) while the ECU controls the exiting engine)

Therefore INTECRIO can be also used to fulfill the methods proposed by ISO26262 as shown below in 7 Appendix.

3.3 Target Prototyping

At latest for target prototyping it is necessary to generate C-code in order to be able to integrate with the rest of the ECU SW (e.g. OS and other basic SW like bus and I/O drivers). At this point it becomes obvious that the INTECRIO concept to integrate always on C-code level also for virtual prototyping and rapid prototyping use case is very helpful as it is possible to use the identical C-code for all prototyping methods. This provides optimal test coverage, as most C-code related errors can be found in early development phases (cost efficient). Target prototyping will identify mainly two issues:

- run time issues as the execution performance of the final target is in general highly reduced compared to virtual/rapid prototyping targets
- compiler/linker/µController issues (e.g. wrong complier optimizations or µController bugs like the DIV problem on the Pentium®)

Creating code for the final target can be done manually (commonly in the C programming language) from the specification. Modern model based development tools, as mentioned above, allow automatic creation of this code from the executable specification.

ISO 26262 requires that in addition to the methods for software unit testing, in chapter 9, section 9.4.6, "*The test environment for software unit testing shall correspond as closely as possible to the target environment.…*".

This requirement of the ISO26262 can best be fulfilled when the software unit under test is executed on the final target. Only here the software can be tested with the same boundary conditions as the final code, therefore allowing a much better verification as a standard processor-in-the-loop test, where still differences to the final hardware set up can exist.

Here, EHOOKs can also provide access to the ECU calculations and, even more, integrate the executable specification on the final ECU target once C-Code is generated from the model (for example, using Simulink's Real Time Workshop®[1] or ASCET). In addition to the capability to patch the ECU software, EHOOKs provides the build environment for the supported targets, allowing an easy on-target prototyping environment. The generated C-code for the newly developed function is compiled for the target, added to the ECU scheduling according to the user settings and merged to the ECU's executable file.

The new variables and parameters of the software to be integrated and tested are added to the ECU's A2L description file (used to give measurement and calibration tools the necessary information to perform their task). After reflashing the ECU with the new files one is able to do measurement and calibration on the new software component in the same way as your can do it for the rest of the ECU.

Furthermore you can use EHOOKS to overwrite ECU variables in order to inject faulty values (ISO26262, ch.10, table 13, 1c) in a much simpler and cost efficient way then generating the errors via hardware-in-the-loop system etc.

The prototype create with EHOOKS is extremely close to the final ECU and therefore very useful for validation and verification purpose as requested by IOS26262.

[1] Trademarks of The MathWorks Inc.

4 Summary

Both presented tools, INTECRIO and EHOOKS, are very helpful to create prototypes on which you can run several tests like back-to-back tests, fault injection tests, etc. as requested by ISO26262. They provide an ECU like/identical environment to test function models and SWCs which make them highly suitable to support validation and verification in various development phases, from MiL/SiL over Rapid Prototyping to On-Target-Prototyping.

5 Reference

1. **ISO26262, Technical Committee ISO/TC.** *ISO/FDIS 26262 - Road vehicles — Functional safety.* Geneva : ISO, 2011. ISO/FDIS 26262:2011(E).
2. IEC. *IEC.* [Online] 6 1, 2011. [Cited: 6 1, 2011.] http://www.iec.ch/functionalsafety/.
3. **Editors: J. Zander, I. Schieferdecker, P.J. Mosterman.** *Model-Based Testing for Embedded Systems.* s.l. : CRC Press, September 9, 2011. 978-1439818459.
4. **Büchner, Frank.** Software-Test - Acht Irrtümer über Code Coverage[1]. *www.elektronikpraxis.vogel.de.* [Online] 6 1, 2011. [Cited: 6 1, 2011.] http://www.elektronikpraxis.vogel.de/themen/embeddedsoftwareengineering/testinstallation/articles/247210/.
5. —. Software-Test - Acht Irrtümer über Code Coverage [2]. *www.elektronikpraxis.vogel.de.* [Online] 6 1, 2011. [Cited: 6 1, 2011.] http://www.elektronikpraxis.vogel.de/themen/embeddedsoftwareengineering/testinstallation/articles/252993/.
6. www.autosar.org. *www.autosar.org.* [Online] 2 22, 2008. [Cited: 6 8, 2011.] http://www.autosar.org/download/AUTOSAR_LayeredSoftwareArchitecture.pdf.
7. **ETAS GmbH.** INTECRIO V3.2 User Manual. *INTECRIO V3.2 User Manual.* Stuttgart : s.n., 2010.

6 Glossary

A2L – ASAM 2 Language from the ASAM organization to describe variables and parameter in an ECU SW
ASIL – automotive safety integrity level (as defined by ISO26262)
DC – decision-coverage
ECU – Electronic Control Unit
E/E/PE – electrical/electronic/programmable electronic
HEX – hexadecimal ASCII file containing the ECU SW and data
HW – hardware (development hardware as well as ECU hardware)
INTECRIO – ETAS SW tool for integration and configuration of SW and HW
INCODIO – Systecs SW tool to ease integration of C-Code into INTECRIO
MC – model-coverage
OSEK-OS – "standard" operating system for ECUs in the automotive industry
SW – software (in this article especially ECU software)
SWC – software component (part/element of the ECU SW)
TCL – Tool Confidence Level
TD – Tool error Detection
TI – tool impact

7 Appendix

The following tables indicate where the tools INTECRIO and EHOOKS can be applied to satisfy the methods recommended by ISO26262. "I" respectively "E" indicate that INTECRIO / EHOOKS can cover most or all aspects of the propose methods. "(I)" / "E" indicate that a part of the aspects of the propose methods are covered. All tables are from (1), part 6.

Chapter 7, Table 6 — Methods for the verification of the software architectural design

Methods		ASIL				
		A	B	C	D	
1a	Walk-through of the design[a]	++	+	o	o	**(I)**
1b	Inspection of the design[a]	+	++	++	++	**(I)**
1c	Simulation of dynamic parts of the design[b]	+	+	+	++	**I**
1d	Prototype generation	o	o	+	++	**I**
1e	Formal verification	o	o	+	+	**(I)**
1f	Control flow analysis[c]	+	+	++	++	**(I)**
1g	Data flow analysis[c]	+	+	++	++	**(I)**
a In the case of model-based development these methods can be applied to the model. b Method 1c requires the usage of executable models for the dynamic parts of the software architecture. c Control and data flow analysis may be limited to safety-related components and their interfaces.						

Chapter 8, Table 9 — Methods for the verification of software unit design and implementation

	Methods	ASIL A	ASIL B	ASIL C	ASIL D	
1a	Walk-through[a]	++	+	o	o	**I**
1b	Inspection[a]	+	++	++	++	**I**
1c	Semi-formal verification	+	+	++	++	
1d	Formal verification	o	o	+	+	**(I)**
1e	Control flow analysis[b, c]	+	+	++	++	**(I)**
1f	Data flow analysis[b, c]	+	+	++	++	**(I)**
1g	Static code analysis	+	++	++	++	
1h	Semantic code analysis[d]	+	+	+	+	

a In the case of model-based software development the software unit specification design and implementation can be verified at the model level.
b Methods 1e and 1f can be applied at the source code level. These methods are applicable both to manual code development and to model-based development.
c Methods 1e and 1f can be part of methods 1d, 1g or 1h.
d Method 1h is used for mathematical analysis of source code by use of an abstract representation of possible values for the variables. For this it is not necessary to translate and execute the source code.

Chapter 9, Table 10 — Methods for software unit testing

	Methods	ASIL A	ASIL B	ASIL C	ASIL D	
1a	Requirements-based test[a]	++	++	++	++	**I**
1b	Interface test	++	++	++	++	**I & E**
1c	Fault injection test[b]	+	+	+	++	**I & E**
1d	Resource usage test[c]	+	+	+	++	**E**
1e	Back-to-back comparison test between model and code, if applicable[d]	+	+	++	++	**I & E**

a The software requirements at the unit level are the basis for this requirements-based test.
b This includes injection of arbitrary faults (e.g. by corrupting values of variables, by introducing code mutations, or by corrupting values of CPU registers).
c 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.
d 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.

Chapter10, Table 13 — Methods for software integration testing

	Methods	ASIL A	ASIL B	ASIL C	ASIL D	
1a	Requirements-based test[a]	++	++	++	++	**I**
1b	Interface test	++	++	++	++	**I & E**
1c	Fault injection test[b]	+	+	++	++	**I & E**
1d	Resource usage test[c, d]	+	+	+	++	**E**
1e	Back-to-back comparison test between model and code, if applicable[e]	+	+	++	++	**I & E**

a The software requirements at the architectural level are the basis for this requirements-based test.
b This includes injection of arbitrary faults (e.g. by corrupting values of variable, by introducing code mutations, or by corrupting values of CPU registers).
c 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 buses) have to be determined.
d 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.
e 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.