-

Significant Quality and Performance Gains through Fully Automated Back-to-Back Testing

Hans J. Holberg
SVP Marketing & Sales, BTC Embedded Systems AG
Buschstr. 1, 26127 Oldenburg, Germany
holberg@btc-es.de

Dr. Udo Brockmeyer
CEO, BTC Embedded Systems AG
Buschstr. 1, 26127 Oldenburg, Germany
brockmeyer@btc-es.de

Abstract: In the meantime it is generally accepted that model driven development is the premise to deliver more embedded functionality in shorter time, in other words with less cost. An additional significant benefit of applying model driven methods for developing embedded software is higher quality of the developed software due to early verification means like model in the loop (MiL[1]) simulation. However, testing of the embedded software is usually still done in a more traditional way, using processes and methods adequate for the software development processes of the 80s and 90s. In fact, there is indeed a significant gap between the high level of productivity of the software engineers, using models to develop the software, and the lower level of productivity of the engineers responsible to perform the testing and quality assurance. An alarming effect is that the quality of the embedded software products decreases, in particular since time-to-market constrains do not relax. In this article we show how to complement model driven development with a model based back-to-back testing approach, and how this leads to significantly improved quality and testing efficiency. The complete model based software verification approach is explained in the context of model driven development using Matlab, Simulink and TargetLink. This approach seamlessly integrates MiL, SiL[2], and PiL[3] testing activities, thereby automating many of the ordinary testing activities. Even more, it shows how the development of the necessary test vectors is highly automated such that the complete verification of the embedded software can be done in much less time than today.

[1] MiL: Model in the Loop. Normally it is a closed-loop system model which consists of the control component plus plant (environmental) model. Here, an open-loop with an automatically generated test harness is used to automatically test the SUT (System under Test).
[2] SiL: Software in the Loop. In contrast to PiL, the real target hardware is replaced by the used host-computer and its ordinary processor. The developed model of the software is only translated into target hardware compatible code. The plant model is replaced by a test driver (automatically generated test harness).
[3] PiL: Processor in the Loop. In contrast to SiL real target hardware (evaluation board) is used to load the application on it for testing. This allows identifying compiler- and processor issues.

1 Field of Application

The described method is based upon an automatic code generation environment[4] and it is seamlessly embedded into a complete development environment for Embedded Software. The automatic test and verification environment[5] is supporting the whole modelling block-set of the automatic code generator. Additionally it is supporting external legacy code which comes from other code generation and even hand written code sources. The currently available solution is supporting any hierarchically developed fixed point and floating point application, such that an extremely high model and code coverage level can be reached by a fully automated approach. This has been successfully proven in series production in the automotive domain during the last 5 years in Germany and Japan.

2 Quality Aspects

As the described method is currently mainly used in the automotive domain, quality aspects can be assessed by using relevant safety standards. Here it is of interest to have a look at the ISO 26262 upcoming standard, which is an adaptation and extension of the currently functional safety standard IEC 61508 especially for functional safety in automotive. The final release of this international standard is planed for 2011 and it is available as a Draft International Standard ("DIS") since mid of 2009. In contrast to the IEC 61508 the ISO 26262 is taking the model-based development process into account. Thus, model-based testing and back-to-back testing between the different development-stages is becoming state-of-the art. ISO 26262 is defining 4 different levels of safety, so called Automotive Safety Integrity Levels (ASIL A, ASIL B, ASIL C and ASIL D). Level A is the lowest and D the highest safety level. For all levels, "Back-to-back tests between Model and Code" are recommended and for level C and D even highly recommended. The quality of the back-to-back tests is determined by using so called coverage criteria. Especially the criteria statement coverage[6], branch coverage[7] and MC/DC coverage[8] are required for the different ASIL levels.

3 Efficiency Aspects

As back-to-back testing on the quality side is state-of-the-art, the next very important question is development and testing efficiency. The different coverage criteria of the specific relevant ASIL Levels are introducing huge additional effort regarding the testing activities of the process. In order to handle this testing complexity automatic approaches are definitely needed to overcome this challenge.

[4] Here: Matlab Simulink/Stateflow (TheMathworks) in combination with the leading automotive code generator TargetLink (dSPACE GmbH) has been used in real serial production projects as standard modelling and code generation environment.
[5] Here: EmbeddedTester from BTC Embedded Systems AG is used. It became a standard test and verification environment for TargetLink users in the automotive domain.
[6] Statement Coverage: Every code statement has been executed during testing at least once
[7] Branch Coverage: Every branch point (decision value) FALSE and TRUE has been taken during testing
[8] Modified Condition Decision Coverage (MC/DC): A set of test vectors, which make every decision TRUE and False while each single condition of that decisions has an independent influence on the value of that decision. A 100% MC/DC coverage guarantees the detection of any failure within a decision of the mode or model.

The first kind of effort complexity is the test creation phase. A method to automate test vector generation to fulfil model and code coverage can tackle this problem. Another kind of problem is to efficiently execute and analyze the huge number of tests. A complete automatic solution is definitely needed in order to prevent from too many manual tasks regarding the testing workflow, and to prevent test errors while performing manual testing. Finally the quality metrics (coverage statistics) needs to be determined from automatically generated test reports. All these arrangements to automate testing can only be efficient, if the techniques are highly integrated within the development and test environments of the software generation tools. This fully automated and integrated solution will be shown in the following sections of this paper.

4 The Reference Work Flow

A reference work flow has been developed in order to define the relationship of model-based development and model-based testing, which shows the paradigm-shift from manual testing on the code implementation level to model-based testing combined with automatic back-to-back testing between model and code levels. In contrast to the traditional development and test process, the new approach focuses the main development and test tasks on the model level and guarantees the correct behaviour transformation by auto code generation in combination with automated structural back-to-back testing. It is widely accepted that testing and debugging on the model level is much easier and cheaper, which makes this approach so attractive. The following figure shows roughly the above explained reference work flow.

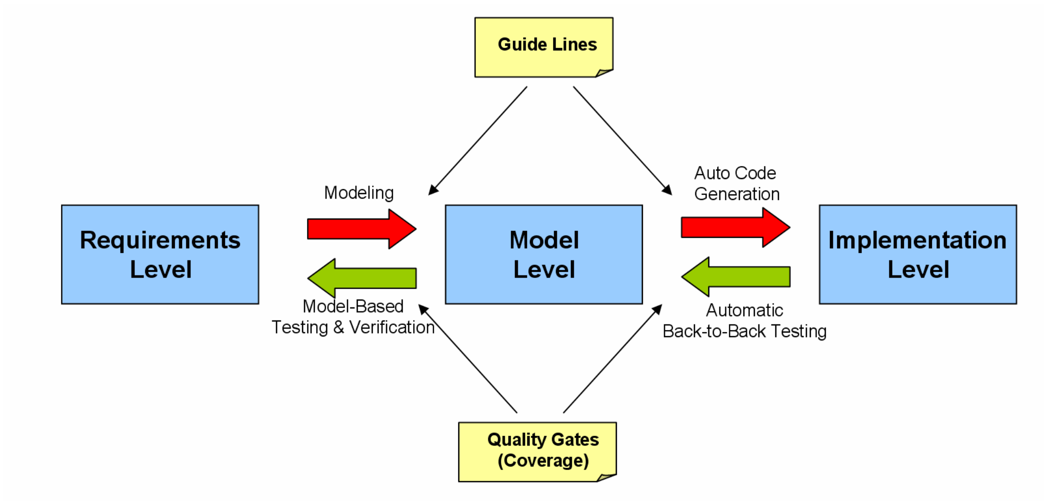


Figure 1: Reference Work Flow[9]

Based on given informal or/and formal requirement specifications executable models can be developed by modelling under certain modelling guide lines. This enables

[9] Published in 2010 by M. Beine (dSPACE GmbH, Paderborn) and Dr. T. Bienmüller (BTC Embedded Systems AG, Oldenburg) under the title: “Addendum to the TargetLink Reference Workflow - Overview and Variations”

model-based testing by using model simulators and even formal verification by using model checkers. The model-based testing is done under well defined quality gates mainly defined by industrial safety standards like the ISO/DIS 26262. If this quality gate is passed, automatic code generation can be used to switch from model level to implementation level. This step(s) have to be verified by using the approach of automatic back-to-back testing, which consists of automatic test generation, automatic test execution including automatic test assessment. The details of this process will be described in the following sections.

5 The Model Based Software Verification Approach

The development and test processes in several industrial domains like automotive and aerospace became more and more model-based. The reason for this trend is obvious as early executable specifications of complete vehicle functions of systems, even including mechatronical components and its network, allows a more efficient development and testing, since not all complex software and hardware design decisions have to be taken upfront. Functions can be developed independent of the final targets, which have a lot of benefits, like IP-Protection, reusability, and better OEM-Supplier-Interfaces. With this approach, the design can be validated very early. Also any single function can be verified against its given requirements. This assures that the whole control system under development is fulfilling the desired system specifications. The usage of automatic code generators for modelling environments tremendously decreases the effort of the implementation phase, but the test effort is still high on the implementation side even if using auto code generation. On the other hand, testing can be lifted-up to the model level by introducing automatic code verification capabilities. The idea of Automatic Code Verification is based on an existing hierarchical model which acts as a behavioural reference (aka “golden device” or “reference model”) for further automatic structural testing on the corresponding code implementation. Basic element of such a testing method is automatic test generation, -execution, -analysis and any kind of reachability analysis. The test generation and the reachability analyses of the Code Verification Environment are performed on the target code itself, which has been generated automatically by the auto code generator. In the first analysis phase all necessary test cases will be generated automatically. Additionally these test cases will be reported to the user within a hyper-linked report, including detailed test information. This report is used as a test center to drive any test activities during the testing process. Beside coverage criteria like condition coverage, decision coverage, CDC[10] and MC/DC-Coverage, also implementation related failure sources like scaling, division-by-zero, saturation and type castings will be taken into account. The intention is an automatic structural comparison with dynamical tests and analyses between the model level(s) which represents the reference level and the implementation level(s) (SiL, PiL and if applicable even HiL[11]).

[10] CDC: Condition Decision Coverage
[11] Hardware in the Loop (HiL): A test method where an embedded system is connected to a HiL-Simulator equipment, which emulates the real environment of the system under test under even real time conditions.

6 Automatic Test Generation and Code-Verification

The automatic test environment is able to find any input stimuli sequence to cover certain coverage criteria. The presented technology is using the automatically generated c-code, in order to represent the software behaviour for further test case generation and code verification analysis. Besides test sequence generation, also unreachable code branches can be identified until an arbitrarily defined analysis depth. Those capabilities are available, because specific algorithms from the area of Formal Methods, which have been used successfully over more than 15 years, are taken into account.

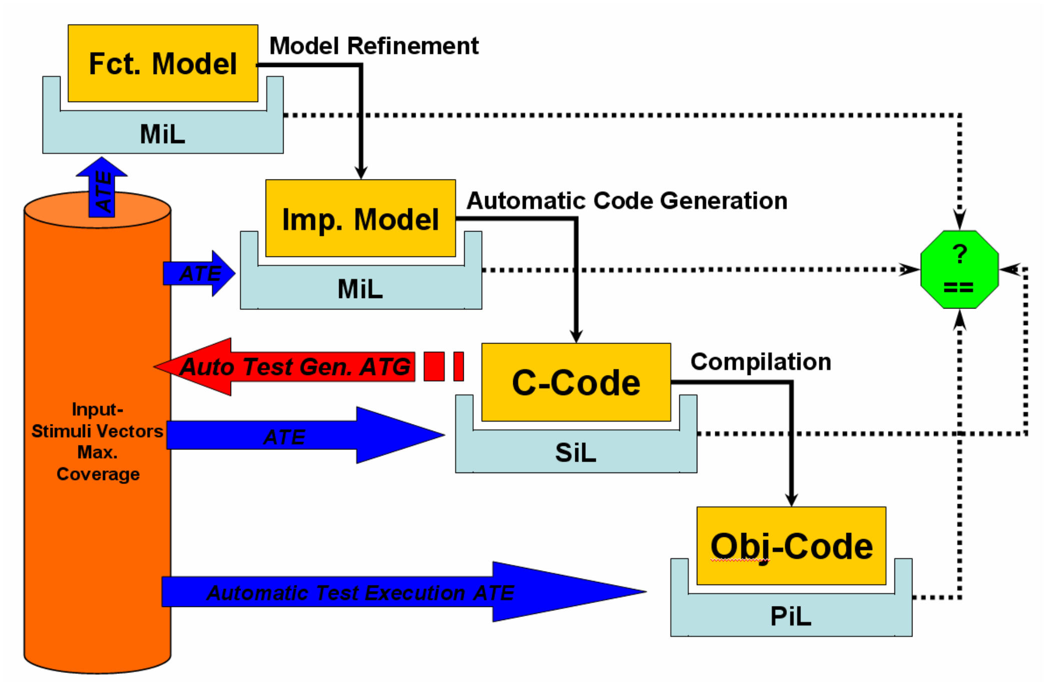


Figure 2: Code Verification Concept

The figure above shows the left side of the V-Process, from target independent Functional Models, via target related Implementation Models to C-Code, and finally the related compiled Object Code running on an evaluation target. Background of the analysis of the verification environment is the c-code.

The automatic test vector generation ("ATG") capabilities of the verification environment is directly using the target c-code to find the right set of input stimuli in order to exhaustively cover code and reference model. Different coverage criteria are measured during test vector generation to maximize the specific desired coverage rates.

The generated or/and imported stimuli vectors are stored within an internal data bank. These vectors are used for execution (ATE) on the different development levels

(Functional Model, Implementation Model, Code and Object Code Levels) to get the needed comparison reference data. The recorded observable variables[12] will finally be compared automatically by using a data stream comparison algorithm. Not acceptable differences are reported by the verification environment, hereby taking user defined tolerances of the specific signals into account.

Due to the tight tool integration of the test and verification environment together with the automatic code generator by using the hierarchical test approach, scalability even up to extremely large industrial applications can be guaranteed.

The following code coverage criteria are currently supported:

- Statement Coverage,
- Condition Coverage
- Decision Coverage
- Switch-Case-Coverage
- Function-Call-Coverage
- Condition Decision Coverage ("CDC")
- Modified Condition / Decision Coverage („MC/DC")

Test cases which are important to check implementation related aspects are the following:

- Division by-Zero,
- Type Range Violations (Over- and Underflows)
- Type-Casting
- Saturation and
- Relational Operations (Fixed-Point vs. Floating-Point),

If the test vector generation algorithms can not completely cover code and model, the verification environment applies formal method techniques to assess the reachability of the missing coverage properties. In contrast to other methods, a so called *handling rate* is introduced by this method. Best practice has shown that a 100% handling rate provides a better metrics than a 100% coverage rate, as 100% coverage under normal conditions never can be reached, for instance, due to safety code around divisions. This shows that a reachability analysis becomes an important element of this testing and verification approach.

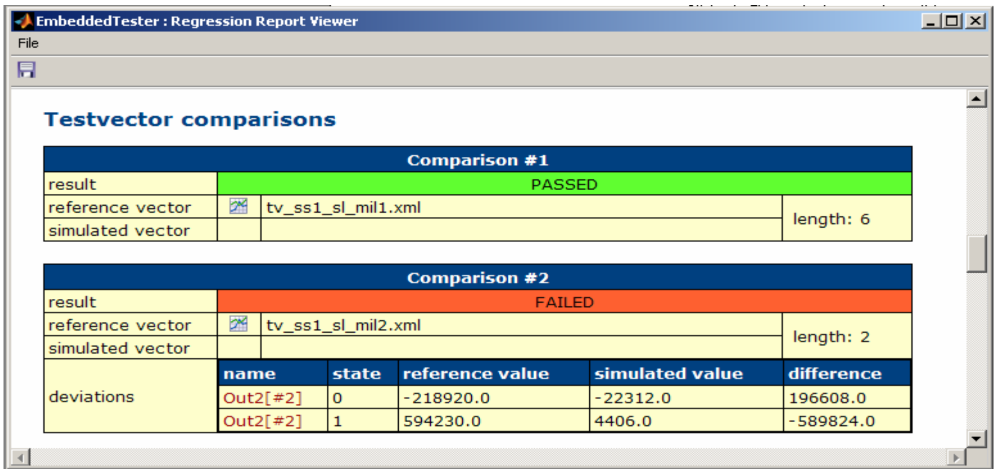
[12] Generally, the verification environment distinguishes between outputs of the "System Under Test" (SUT) and the observable internal signals, which can be used for testing purposes. If the user in only interested in output signals of the SUT, it is called "Black Box Testing". If internal signals are needed for diagnosis the "Grey Box Testing" mode is used.

7 Automatic Test Execution

Due to the consequent hierarchical approach, which guarantees scalability over the industrial sized applications, the automatically generated stimuli sequences can be executed on the different execution levels of the corresponding hierarchy entity (interface) for recording the behavioural reactions of the particular function/system. The needed test harness generation is generated fully automatically by the verification environment without any user intervention and effort. It also guarantees that the system under test (target code) is not touched or modified while testing. This approach completes the auto-generated stimuli-vectors into real test vectors consisting of input sequences and its calculated corresponding output (observable) expectation values.

8 Automatic Test Evaluation

In back-to-back testing mode, the verification environment fully automatically compares the executed test cases, including reference (output) values, on all levels (MiL, SiL and PiL), and shows the differences in automatically generated reports. Tolerances can also be defined to fine-tune the automatic comparison of data streams. Fixed-point versus floating-point aspects are in particular addressed during automatic test evaluation.



Testvector comparisons

Comparison #1			
result	PASSED		
reference vector		tv_ss1_sl_mil1.xml	length: 6
simulated vector			

Comparison #2					
result	FAILED				
reference vector		tv_ss1_sl_mil2.xml			length: 2
simulated vector					
deviations	name	state	reference value	simulated value	difference
	Out2[#2]	0	-218920.0	-22312.0	196608.0
	Out2[#2]	1	594230.0	4406.0	-589824.0

Figure 3: Test Evaluation Report

The figure above shows a test evaluation report which summarizes all test results. It indicates if tests yielded not expected values regarding the user defined tolerance range. This report is also hyper-linked to the test manger straight pointing to the relevant test vectors to enable highly automated and efficient debugging.

9 Debugging Support

If differences between the execution levels are discovered, finding/fixing the source of an error becomes an issue. The code verification environment supports users with linked coverage reports and dedicated debugging facilities. Automatically generated hyper-linked reports show differences between target code and the corresponding reference model(s), if a user-defined deviation/tolerance is violated. With a single mouse click, the user is able to jump to the corresponding, relevant code or/and model part in order to analyze the reason for unacceptable differences. This advantage can be achieved only by a close integration of the modelling, code generation and verification environment. Finally, test vectors can be debugged step-by-step on MiL and SiL levels (automatic export of e.g. Visual C-Debugging-Project). This integration significantly decreases debugging setup and execution efforts.

10 Open Import- and Export Interfaces

The Code Verification Environment supports importing and exporting test vectors into and from numerous file formats, such as XML, MAT, XLS, CSV, CTE and others. It enables to easily use new and existing test sets from various sources. After importing test cases, the verification environment shows their achieved code coverage ratio. Test cases can also be reused, which have been automatically generated by requirements based test and verification environment such as model checkers etc. Furthermore, test vectors can also be defined interactively by the user on the basis of requirements by using an integrated or plugged-in test authoring system. All managed test cases also can be exported to be reused in subsequent testing stages, such as in HiL-Testing-Environments.

11 Practical Experiences

Experiences from the field of customers demonstrated that the presented automatic back-to-back testing method enhanced with automatic test vector generation saves 80-90% of the test creation, execution and analysis efforts compared to conventional manual approaches.

Additionally it proved that the achieved quality level can be significantly improved as demanded by the upcoming ISO DIS 26262 standard. In particular it is very helpful in this process to measure “achieved test quality” by using well-known coverage criteria such as Branch coverage or MC/DC coverage. Coverage rates can easily be increased by more than 25% by using automatic methods.

Another very important advantage of this highly integrated technology is the almost fully automated debugging support. First user feedbacks have shown a time saving of at least 50% compared with a fully manual approach.

12 Conclusion

The presented model based back-to-back testing approach is a quantum jump in the direction of more efficient testing, since the total testing effort can be tremendously minimized, while the quality of the product under development can be significantly increased. This has been proven during the last 4 years in serial production in the automotive industry.

The key for an efficient use of this fully automatic model based software verification and testing approach is a close, hand-in-hand integration with high efficient and de-facto standard tools for automatic code-generation in tight connection with the real implementation level and the modelling environment, all this of course fully embedded in the model-based development process.

Due to the re-use of formal methods technology on the code-level, providing answers to the “issues of completeness”, any desired quality level, mainly derived from industrial standards like ISO 26262, IEC 61508 or DO-178b, can be reached without over-proportional manual test effort. A mandatory key element is the tight combination of the test execution with coverage measurement technology in order to achieve the “testing quality” at any point in time during development and testing.