

Automatic Test Case Generation for Safety-Related Embedded Systems

Peter Dämon

Magna Steyr Fahrzeugtechnik AG & Co KG

Copyright © 2008 SAE International

ABSTRACT

Software testing for embedded systems is based on a labor-intensive manual process, which often fails to provide the required level of quality. Nevertheless, faults within software systems, especially those which are relevant for safety-related applications (e.g., Steer-by-Wire, Break-by-Wire), must be detected to prevent the occurrence of a catastrophe. Considering upcoming technologies such as X-By-Wire systems, future software for embedded systems will become more complex and a higher level of quality has to be achieved. This will lead to a significantly increased testing effort. Thus, a systematic method for automated software testing is highly desirable.

INTRODUCTION

The number and complexity of automotive systems is constantly increasing. Premium cars contain up to 70 electronic control units (ECUs). Furthermore, it is a challenge in future automotive systems to realize complex safety-related functions based on an electronic system without relying on mechanical or hydraulic backup. These so called X-By-Wire systems, e.g., steering, braking, powertrain and suspension control will provide a big potential in increasing active and passive safety. As a result of this trend a high degree of software quality must be achieved in order to ensure functional safety. To avoid an explosion of the testing effort improved testing methods are required.

This paper describes a testing framework, which aims to automate software testing using a model-based, requirement-based and formal approach. It is integrated in a state-of-the-art model-based software development environment which comprises the tools Matlab®, Simulink®, Stateflow® and TargetLink®.

SAFETY STANDARD IEC 61508 - The prerequisites for the quality of safety-related systems are usually defined

in safety standards. The safety standard IEC 61508 [1, 2] covers all industrial fields and all aspects of the functional safety of electrical/electronic/programmable electronic safety-related systems. Hence, the standard IEC 61508 is widely accepted in the automotive domain. It provides a versatile set of development and testing techniques which are, depending on the safety integrity level (SIL), more or less recommended.

Safety Integrity Level	Damage
SIL1	Injury
SIL2	Death to one
SIL3	Death to few
SIL4	Catastrophe

Table 1 Safety Integrity Levels

Safety-related systems in the automotive domain are considered to be classified between SIL1 and SIL3. The introduced testing framework is orientated to this standard and fulfills the following safety-related aspects:

Automated testing - A main goal of a safety-related software development process is the avoidance of human failures. This can be achieved with the aid of automation. For this reason the test cases are automatically generated, applied, evaluated, documented, and administrated by the testing framework. Besides building a formal model of the Unit-Under-Test (UUT) manually the model can be extracted automatically out of the C source code [10].

Systematic testing - In general and especially for safety-related software, the overall testing activity must be systematically planned rather than performing it in an ad-hoc manner. Thus, systematic testing is a basic requirement for a safety-related software development process (IEC 61508-3 7.3). It subsumes all testing

activities that examine the behavior of the UUT under specific conditions with particular expected results. Accordingly, the approach of the described testing framework to systematic testing is to systematically create test cases based on requirements.

Structure-based testing - The basic idea of structure-based testing is to find tests which execute certain parts of the software. Structured-based testing is recommended for SIL1-SIL2 and highly recommended for SIL3-SIL4 (IEC 61508-3 B 2.6). To comply with this subject the described testing framework aims to meet concrete coverage criteria.

Requirement-based testing - The IEC 61508 standard postulates that it has to be satisfactorily shown that each software module covers all safety requirements while not containing any dead code. Consequently, it is necessary that the test cases are related to the requirements. (IEC 61508-3 7.4.7.2).

Modeling - A model is usually an abstract, partial presentation of the system's desired behavior. It is built before or parallel to the development process of the UUT (IEC 61508-3 A 7.2). To model the system behavior the automaton-based system specification language SMV [11] is currently used which comprises following features:

- Support of discrete time
- Suitable to model finite state machines (FSM)
- Can be processed by a model checker
- Provides features to construct parametric models from which concrete models can be instantiated.
- Comprehensive specifications
- Supports the modeling of the system with multiple abstraction levels
- On the whole, the language is expressive enough to model embedded software for the automotive domain

Formal methods - Sophisticated and mature testing techniques using formal methods are of common use in highly safety-related areas, for instance transportation, avionics, or aerospace. In the automotive domain formal methods for testing are only starting to gain ground. However, formal methods are recommended for SIL 2 and highly recommended for SIL3-SIL4 (IEC 61508-3 A 4.1c) [1]. For that reason, a formal model of the SUT is built by using the formal modeling language SMV. Furthermore the requirements are formally specified in order to express the specification unambiguously and consistently. For this purpose the formal specification language PSL/Sugar [12] is currently used which comprises following features:

- Has been in use for more than ten years
- IEEE standard since the end of 2005
- Can be processed by a model checker
- Tools are being developed
- Supports the specification of linear time constraints
- Supports the specification of computational tree constraints

Dynamic testing - Contrary to static testing (where the software is not actually used), dynamic testing means that concrete test cases are executed on the SUT. Dynamic testing is recommended for SIL1 and highly recommended for SIL2-SIL4. (IEC 61508-3 A 5.2) [1]. As the introduced testing framework generates, applies, evaluates, documents and administrates test cases, it conforms to this subject.

Traceability - In software development, the term traceability refers to the ability to link the requirements set forth at the beginning of a project, to the resulting software, the associated test cases and all related documents. Traceability is stipulated by all modern software development standards. That means, that the testing framework needs to link the requirements to the related test cases and to the resulting test reports.

MAIN SECTION

APPLICATION DOMAIN - MAGNA Steyr Fahrzeugtechnik AG & Co KG (MSF) has adopted a model-based software process for the development of application software. The advantage of modeling and simulating application software which allows for rapid-control-prototyping, automated code generation and advanced test-techniques such as Model-in-the-Loop (MIL), Software-in-the-Loop (SIL) and Processor-in-the-Loop (PIL) are well known. However, there are some drawbacks. Well implemented development methods for manual written code (C, C++, Java, etc.), such as structural programming, software modularity or configuration management, need to be newly defined and implemented. Structural programming and software modularity is an important concept in all modern software engineering methodologies and is highly recommended for all safety integrity levels (IEC 61508-3 7.4.5) [1]. Unfortunately, the tools which are currently available on the market don't feature all necessary properties to fully support a model-based development process. Consequently, MSF is actively developing enhancements to the Matlab® environment. In order to realize software modularity and structural programming a tool has been developed (Model Composer) which allows for importing/exporting different variants of modules. Further more, a configuration management has been realized using the tool MKS Source Integrity®.

Currently, MSF further enhances the Matlab® environment to implement a testing framework which allows for the automation of important steps within the test process and for the integration of automated test case generation.

As mentioned earlier, the application software is preferably developed model-based at MSF. The application software is the functional part of the ECU software which usually consists of two main parts:

Control algorithm - The control algorithm describes the functional behavior of the ECU outputs as a function of

the inputs. The inputs to the algorithm are the sensor signals. The outputs from the algorithm are the control signals sent to the appropriate actuators in order to attain a desired operating condition. The control and data flow of an control algorithm is mostly rather simple. However, due to the closed-loop characteristics of a control system, the history of the whole system's behavior is considered in the current calculation. This leads to an extremely huge state space which makes the application of formal methods not suitable. The main challenge in designing control algorithms lies in the optimization of the transient response and the stability. These subjects are not in the scope of the introduced testing framework.

Monitoring and diagnostic software - The monitoring and diagnostic software checks the plausibility of incoming signals and detects possible functional misbehaviors of the control algorithm. In case of a detected misbehavior or non-plausible signals, the software needs to react in an appropriate way. For instance to set the system into a pre-defined safe state. Following this approach, the monitoring and diagnostic software must primarily ensure the functional safety of the software system.

The functional behavior of a monitoring and diagnostic software system is typically state-orientated. Hence, models of such software are usually realized as state chart models which represent the behavior of the underlying system in terms of states and transitions.

The monitoring and diagnostic software turned out to be the bigger part of the application software and today's size and complexity of such models makes it almost impossible to manually verify all states and transitions within an adequate time. Thus there is strong need for incorporating novel verification techniques for this kind of software.

The introduced formal methods are well suited for state oriented software which comprises a complex control flow. Hence, the focus of the introduced testing framework lies on the verification of monitoring and diagnostic software.

Nevertheless, the limits of the applicability of formal methods are given by the difficulty to use them for large software. Thus, the testing framework aims to verify software modules. Consequently, a fundamental prerequisite for the application of automated test case generation using formal methods is a software development process which supports software modularity.

CONCEPT OF TESTING FRAMEWORK – A prerequisite for automated test case generation is the availability of a formal behavioral model of the software and the related formal requirements. Consequently, a first vital step is to formalize the verbal formulated requirements and to create a formal model of the software. The model can either be manually created or automatically extracted out of the source code. The

requirements need to be manually formalized based on the available verbal formulated requirements.

A big advantage of the formal approach is, that the behavior of the software modules is described unambiguously and consistently. This helps to uncover a lack of clarity in verbal formulated requirements. In any case, considering a safety-related software development, the unambiguity and consistency of requirements are basic requirements.

The introduced testing framework comprises two verification steps :

1. Formal verification [3-4]
2. Verification by test cases. This includes test case generation, application, evaluation, documentation and administration

Figure 1 shows the concept of the testing framework.

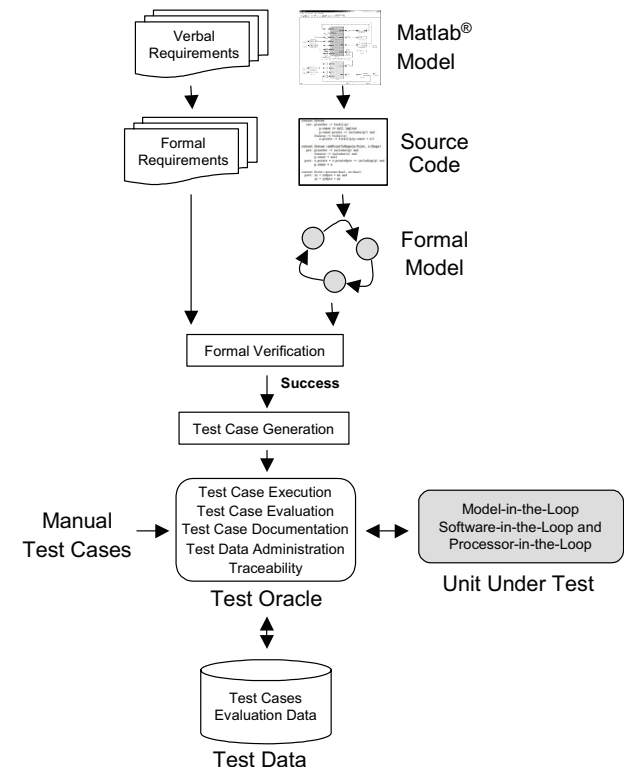


Figure 1 Concept of Testing Framework

Both verification steps are based on model-checking which was initially invented for formal verification purposes only. However, further techniques were developed, which allow for automated test-case generation [5-9]. Currently, the model checker NuSMV [11] is used.

Formal verification – A model-checker takes as input a formal model and a requirement which is related to the