

Model-Based Real-Time Testing of Embedded Automotive Systems

Pawel Skruch and Gabriel Buchala
Delphi Automotive

ABSTRACT

The paper presents a model-based approach to testing embedded automotive software systems in a real-time. Model-based testing approach relates to a process of creating test artifacts using various kinds of models. Real-time testing involves the use of a real-time environment to implement test application. Engineers shall use real-time testing techniques to achieve greater reliability and/or determinism in a test system. The paper contains an instruction how to achieve these objectives by proper definition, implementation, execution, and evaluation of test cases. The test cases are defined and implemented in a modeling environment. The execution and evaluation of test results is made in a real-time machine. The paper is concluded with results obtained from the initial deployment of the approach on a large scale in production stream projects.

CITATION: Skruch, P. and Buchala, G., "Model-Based Real-Time Testing of Embedded Automotive Systems," *SAE Int. J. Passeng. Cars – Electron. Electr. Syst.* 7(2):2014, doi:10.4271/2014-01-0188.

INTRODUCTION

The extensive use of electronics and software within the last decades has revolutionized the implementation and scope of modern embedded systems in the automotive industry. The embedded systems have become increasingly sophisticated and their software content has grown rapidly. Model-driven development (MDD) and model-based testing (MBT) methodologies have become the preferred approaches for the development of such systems. MDD describes a software development approach in which models of software systems are used for application design and implementation. MBT relates to a process of creating test artifacts from various kinds of models by application of a number of sophisticated methods. The results of the study [2] show that with the right (i.e., correctly applied) approach MDD and MBT can bring significant cost savings and also improvement in the quality of the final system.

Real-time testing involves the use of a real-time environment to evaluate the system at its normal operating frequency, speed, or timing. Engineers shall use real-time testing techniques to achieve greater reliability and/or determinism in a test system. Real-time testing shall enable to process all signals simultaneously (i.e., respecting precise timing requirements) and evaluate test results online during test execution. Online test evaluation provides results immediately (once a test case is executed) and without test engineers spending additional time and effort on the offline analysis of the logs. Additionally, an online evaluation testing strategy can be adapted depending on the results as they occur to make the tests more efficient.

The paper presents an integrated model-based approach for testing embedded automotive software systems in a real-time. The presented approach concerns mainly black-box testing and includes definition and implementation of test cases in a modeling environment and execution and evaluation of test results in a real-time hardware. The paper is organized as follows. In the next section, possible scenarios of using models in system development process are reminded. Following section deals with a representation of the system under test (SUT) under which the presented approach is based on. The sections called 'Test Notation' and 'Test Implementation' describe in details definition and implementation aspects of test cases in MATLAB®/Simulink® environment. The next sections called 'Test Evaluation' and 'Test Execution' are devoted to test execution aspects in dSPACE™ real-time hardware. Conclusions complete the paper.

SYSTEM DEVELOPMENT USING MODEL-BASED APPROACH

The model-based design process, which includes both the code development and test development processes, is detailed in [Figure 1](#) and [Table 1](#). In this process, software code development and test development are performed concurrently. Very important for those processes are system requirements which form an input for both software engineering and test engineering activities.

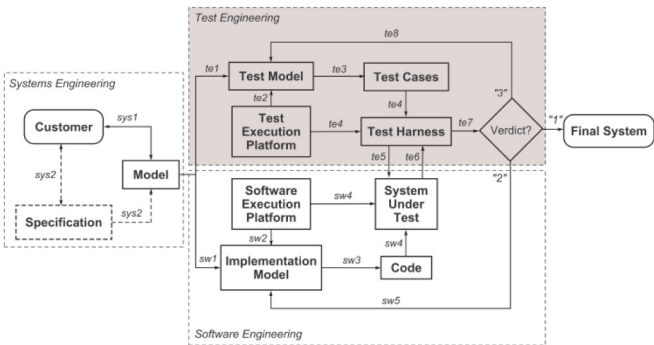


Figure 1. Overview of model-based development and testing processes.

Table 1. Activities in model-based development and testing processes.

Competency	Event	Description
Systems	sys1	elicit customer requirements and express them in the form of a model
	sys2	build a model based on written specification (optional)
	sys3	elicit customer requirements and express them in the form of written specification
Software	sw1	create an implementation model
	sw2	add to the model constraints imposed by software execution platform
	sw3	(auto)-coding
	sw4	integrate the code with execution platform
	sw5	correct the model
Test	te1	create a test model
	te2	add to the model constraints imposed by test execution platform
	te3	(auto)-generate test cases
	te4	create test suite and integrate it with test execution engine
	te5	execute the test suites
	te6	generate test reports
	te7	evaluate test results
	te8	correct the model
	"1"	system ok
	"2"	software error
	"3"	testing error

Models can be used in engineering disciplines to specify the system’s behavior in a clear and unambiguous form. In such case, the models are built based on the requirements defined by customer and often delivered in the form of written specification. Before the requirements are modeled or specified they must be gathered through an elicitation process. Requirements elicitation is a part of the requirement engineering process that is in scope of systems engineering competency. The model of the developed system can reside at different levels of abstraction and it is usually created for different purposes. An implementation model is designed to meet the requirements from a software engineering point of view and it is often used together with a code generator. The implementation model takes into consideration the various target system implementation constraints such as real-time operating system, scheduling details, signal representations, fixed-point vs. floating-point

number specification, etc. Models used to generate test cases are called test models. Such models usually capture the intended behavior of the SUT and contain test implementation and execution constraints imposed by test execution platform. Once the code is compiled and integrated with other software and hardware components, the entire system is being tested. Testing process is conducted through the test harness that allows executing of test cases and generating associated test reports. The primary goals of the testing process are to find defects and to show the system’s compliance to its requirements. If the tests fail, then either the implementation model needs to be redesigned (because of software error) or the test model has to be changed (because of testing error). The system is not released until all tests pass.

The model can be also used as an oracle that is a source to determine expected results to compare with the actual result of the SUT [1]. The approach (Figure 2) stipulates that the same inputs are applied to both the SUT and to the model. The signals are physical in case of the SUT and virtual in case of the model. The judgment whether the result of a test is in conformance with the model is delegated to a test comparator. The test comparator is usually a tool that compares the actual output produced by the SUT with the expected output produced by the model.

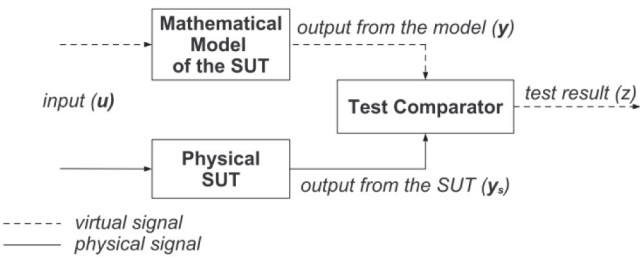


Figure 2. Concept of testing with a model as an oracle.

In this paper, models are used to create test environment and test harness where the tests cases can be defined and implemented and then executed and evaluated in a real-time execution platform.

SYSTEM REPRESENTATION

We consider the SUT as a set of inputs $u(t) \in U \subset \mathbb{R}^r$, state variables $x(t) \in X \subset \mathbb{R}^n$, and outputs $y(t) \in Y \subset \mathbb{R}^m$ that are expressed as vectors (Figure 3). Here, U is called the input state space, X is called the internal state space, Y is the output state space, $\mathbb{R}^r, \mathbb{R}^n, \mathbb{R}^m$ are real vector spaces of column vectors, the independent variable $t > 0$ is time, $x(0) \in \mathbb{R}^n$ is the given initial condition r, n, m are positive integers that determine numbers of inputs, state variables, and outputs, respectively. The relationship between input, state, and output variables is determined by system requirements and can have different forms (e.g., logical formulas, state machines, algebraic equations, differential equations, etc.).

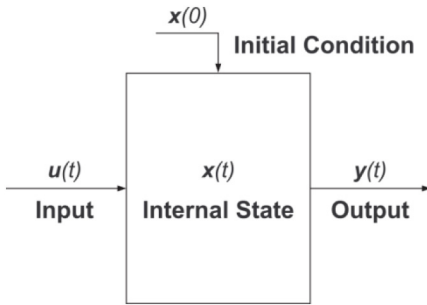


Figure 3. Input/state/output representation of the SUT.

The system representation in the form of input/state/output notation (called also state space notation) provides a convenient, compact, and universal way to model and analyze systems with multiple inputs and outputs

TEST NOTATION

The system specification in the form of an input/state/output representation can be used to describe tests in the way to be independent from test methods, test implementation, and test execution environment. The industry standard [4] defines a test case as a set of inputs, execution preconditions, and expected outcomes developed for a particular objective, such as to exercise a particular program path or to verify compliance with a specific requirement. Following this definition, a single test case $T_{\text{case}}^{(j)}$ can be specified as

$$T_{\text{case}}^{(j)} = \{T^{(j)}, x_0^{(j)}, u^{(j)}(\cdot), y^{(j)}(\cdot)\}, \quad (1)$$

in case of black-box testing [3], or

$$T_{\text{case}}^{(j)} = \{T^{(j)}, x_0^{(j)}, u^{(j)}(\cdot), x^{(j)}(\cdot), y^{(j)}(\cdot)\}, \quad (2)$$

in case of grey-box testing [6]. Here, $u^{(j)}(\cdot)$ is an input function applied to the tested system, $x^{(j)}(\cdot)$ is an expected state function, $y^{(j)}(\cdot)$ is an expected output function within the execution time window $[0, T^0]$ when the system starts from the initial state $x_0^{(j)}$, $j = 1, 2, \dots, N$ is a label to indicate different test cases. A collection of one or more test cases forms a test suite $T_{\text{suite}} = \{T_{\text{case}}^{(1)}, T_{\text{case}}^{(2)}, \dots, T_{\text{case}}^{(N)}\}$.

The test cases (1) and (2) are defined on the basis of continuous-time signals. In the implementation these signals have to be approximated by discrete-time signals. In the presented approach, a continuous-time signal is approximated by a set of discrete points with linear interpolation between them. Figures 4 and 5 illustrate this implementation for digital and analog signals.

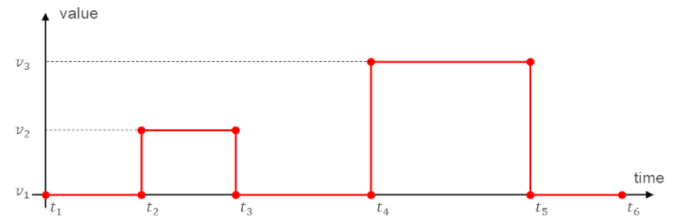


Figure 4. Implemented representation of digital signals.

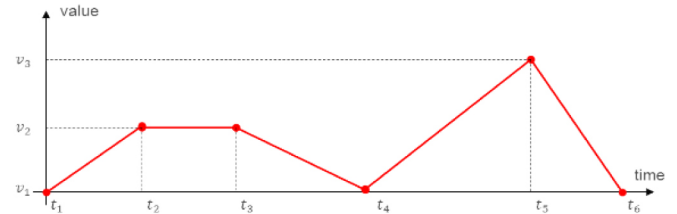


Figure 5. Implemented representation of analog signals.

Formally, these representations can be described using vectors for time and value coordinates, that is,

$$s_{\text{time}} = [t_1 \ t_2 \ t_2 \ t_3 \ t_3 \ t_4 \ t_4 \ t_5 \ t_5 \ t_6], \quad (3)$$

$$s_{\text{value}} = [v_1 \ v_1 \ v_2 \ v_2 \ v_1 \ v_1 \ v_2 \ v_2 \ v_1 \ v_1], \quad (4)$$

in case of the digital signals plotted on Figure 4, and

$$s_{\text{time}} = [t_1 \ t_2 \ t_3 \ t_4 \ t_5 \ t_6], \quad (5)$$

$$s_{\text{value}} = [v_1 \ v_2 \ v_2 \ v_1 \ v_3 \ v_1], \quad (6)$$

in case of the analog signal plotted on Figure 5. The implemented representation of a continuous-time signal is then characterized by a pair $(s_{\text{time}}, s_{\text{value}})$, where s_{time} stands for the time coordinate and s_{value} stands for the value coordinate of the represented signal.

It shall be noticed that the approximation of the continuous-time functions in the form (3), (4) and (5), (6) is perfectly and fully supported by MATLAB®/Simulink® environment.

TEST EVALUATION

Evaluation and test comparison mechanism are necessary parts of any successful test design as they are used for determining whether a test passes or fails. The comparison mechanism is often named in software engineering as test comparator [5]. The test comparator (see also Figure 2) is usually a combination of software and hardware components as its intention is to compare signals of different nature (i.e., physical vs. virtual). It shall be underlined that the comparison of signals significantly differs from the comparison of single values and it can be realized either in time or value domain. Following the concept presented in Figure 2, the test