

# Automatic Test Sequences Generation from Models

Robert Busser, Mark Blackburn, Aaron Nauman

Software Productivity Consortium/T-VEC  
2214 Rock Hill Road, Herndon, VA 20170

**Abstract** - This paper discusses recent advancements of the test generation system to produce test sequences for testing dynamic systems from design modeling systems such as Mathworks' Simulink® and National Instruments' MATRIXx®. Test sequences support test generation for systems that are modeled using constructs that support feedback, such as integrators or time delays, which are common in control system models. Test sequences can address the logic paths, and computation testing in the software as well as dynamic aspects of systems response.

## I. OVERVIEW

Requirement and design-based models are used in aircraft and automotive software-system development where high reliability is demanded. Some rigorous modeling approaches support simulation and code generation, but have limited support for automated test generation. To address this need, the Test Automation Framework (TAF) approach for model-based analysis and test automation was developed [1]. TAF integrates various government and commercially available model development and test generation tools to support defect prevention and automated testing of systems and software. TAF supports modeling methods that focus on representing requirements, like the Software Cost Reduction (SCR) method, as well as methods that focus on representing design information, like Simulink or MATRIXx, which supports control system modeling.

Model translators convert requirement-based or design-based models into a form where T-VEC, the test generation component of TAF, can produce tests vectors. Test vectors include inputs as well as the expected outputs with requirement-to-test traceability information. T-VEC also supports test driver generation, requirement test coverage analysis, and test results checking and reporting. The test driver mappings and test vectors are inputs to the test driver generator, which produces test drivers that are then executed against the implemented system during test execution.

This paper focuses on automatic test generation of test sequences to support dynamic systems testing. While TAF/T-VEC has been used primarily for verifying the static response of systems under test, this paper describes how automated generation of test sequences are suitable for verifying the performance of systems that also exhibit dynamic response. The T-VEC mechanisms are configured

to include the semantics associated with multiple execution cycles of a given system. T-VEC has been used to automatically determine input values for state memory variables that characterize such systems. It can also use the state-space solver capabilities of T-VEC to assist in synthesizing important gain coefficients the designs require.

## A. Background

The core capabilities of this approach were developed in the late 1980s and proven through use in support of FAA certifications for flight critical avionics systems [2]. The approach supports requirement-based test coverage mandated by the FAA with significant life cycle cost savings [3; 4; 5].

The approach and tools have been used for modeling and testing system, software integration, software unit, and hardware/software integration functionality. It has been applied to critical applications in medical and aerospace, supporting automated test driver generation in most languages (e.g., C, C++, Java, Ada, Perl, PL/I, SQL), as well as proprietary languages, and test environments.

## B. Concepts and Definitions

Logic design is often categorized as combinatorial and sequential. Combinatorial logic is comprised strictly of stateless logic operators (e.g., AND, OR, NOT). The output values produced by combinatorial logic are expected to be the same for the same set of inputs values. Systems based on combinatorial logic maintain no memory of input values or of computation results from previous execution cycles and exhibit a static response to the values of its input variables. Sequential logic includes one or more components that maintain knowledge of the input values from previous execution cycles, such as flip-flops or time-delay operators, and whose output values depend not only on the current inputs but also on historically maintained knowledge. The output response of a sequential logic system can vary from cycle to cycle in reaction to the history of inputs within the system. Systems of this type exhibit a dynamic response in addition to their static response characteristics.

## C. Dynamic Response Models

Design models used for simulation and automatic code generation often include input-to-output relationships involving multiple cycles of execution. This is due to the

use of primitive operators that have state memory feedback semantics of sequential logic designs (e.g., TimeDelay in Simulink). These types of operators are often used to design digital signal processing applications such as signal frequency sensitive filters and feedback-loop control laws for digital control applications. Such applications exhibit a dynamic response to their input signal values.

When an application's design includes dynamic response characteristics, it is often difficult to predict the expected output values for a given set of input values when only considering a single cycle's inputs. Consequently, verifying the correct operation of such a design is non-trivial and compiling verification evidence of proper functionality with traditional software testing approaches can be problematic. However, verification evidence is often required by customers and certifying agencies, such as the FAA.

#### D. Traditional versus Dynamic Testing

Traditional software testing approaches, generally centered around developing and applying suites of test cases, where each test case is comprised of a set of input values and an expected output value, are geared towards verifying a system's static response. The system under test (SUT) is initialized with input values, is executed from a specific start point to specific end point in the application's instruction space, the actual value of one or more output variables is extracted and compared to the expected output values, and the results of these comparisons determine the pass or fail status of the test. Each test must examine a single input-to-output execution cycle, essentially one state transition of the overall system. Tests of this type are expected to be repeatable any number of times in sequence, and the same input values are expected to result in the same output values. However, the use of operators with state memory semantics can render such single state transition test cases non-repeatable. Each successive execution of the test can result in unique output results. Such a testing approach is inadequate for verifying the time-wise non-linear or state-machine-based characteristics found in such models.

It is possible to test a SUT's dynamic response using the test case approach by modeling state memory variables as additional input variables. However, it can be difficult to determine what values these state memory inputs should be for a given test case because they depend directly on the history of inputs. The mechanisms providing state-memory semantics, as well as the mathematical relationships characterizing system response in terms of inputs and state memory are complex. The requirements governing dynamic response are often expressed in terms of output value tendencies, such as *rise time*, *over shoot*, and *settling time* rather than functional value mappings between a single input value set and an associated output value.

Requirements describing a system's static response can be formally expressed in terms of pre and post-condition

pairs. The pre-condition characterizes the system states under which the post-condition's input-values-to-output-value mapping must hold. The requirements governing a given output is said to be "complete" if there is at least one pre and post condition pair describing the value of the output in terms of input values for all points in time and modes of system operation. They are said to be "consistent" if there is at most one such pre and post-condition pair for a given output variable for any given point in time. A set of test cases is associated with a complete and consistent set of pre and post-condition pairs that can be shown to produce MCDC-complete requirements-based tests.

#### E. Test Sequence Vectors

To verify that an implementation of a model correctly provides required responses to a step input signal, one needs to test the implementation's response over a period of time through multiple execution cycles. Consequently, test cases that include an association between a single set of input values and a single expected output value cannot adequately verify such performance. A new concept, **test sequence vectors** (TSVs), is required for specification-based test generation.

Informally, a TSV is a test specification that includes the input values for a sequence of execution cycles of the system under tested. A TSV includes values for each independent input variable for each invocation of the model. It also contains initial condition values for the closed-loop feedback variables utilized by the first invocation in the sequence. Lastly, a TSV includes expected output values for each individual system invocation in the sequence, as well as the final expected output values for the overall sequence.

#### F. Future Work

Future work will describe how the tools are used to automatically generate test sequence vectors and will describe how the sequence vectors produced provide sufficient information for thoroughly testing models.

### REFERENCES

- [1] Blackburn, M.R., Using Models for Test Generation and Analysis, Digital Avionics System Conference, October, 1998.
- [2] Blackburn, M.R., R.D. Busser, T-VEC: A Tool for Developing Critical System. Eleventh International Conference on Computer Assurance, June, 1996.
- [3] Statezni, D., Industrial Application of Model-Based Testing, 16th International Conference and Exposition on Testing Computer Software, June 1999.
- [4] Statezni, D. Test Automation Framework, State-based and Signal Flow Examples, Twelfth Annual Software Technology Conference, May 2000.
- [5] Safford, E.L. Test Automation Framework, State-based and Signal Flow Examples, Twelfth Annual Software Technology Conference, May 2000.