A Systematic Approach to the Test of Combined HW/SW Systems

Alexander Krupp, Wolfgang Müller
Paderborn University / C-LAB, Paderborn, Germany

***Abstract*—Today we can identify a big gap between requirement specification and the generation of test environments. This article extends the Classification Tree Method for Embedded Systems (CTM/ES) to fill this gap by new concepts for the precise specification of stimuli for operational ranges of continuous control systems. It introduces novel means for continuous acceptance criteria definition and for functional coverage definition.**

I. Introduction

Current system verification approaches exhibit a major gap between requirements and formal property definition. However, today widely accepted means for natural language formalization and formal property definition do not exist as a target for automated test execution with support of verification planning. The Classification Tree Method for Embedded Systems (CTM/ES) [6] has been introduced for the test definition of automotive systems to fill this gap. However, it does not sufficiently cover all aspects for automatic testbench generation.

This article presents an approach for the definition of a functional verification plan for combined HW/SW systems based CTM/ES. Our approach closes the methodical gap between requirements specifications and test definition by the introduction of an enhanced CTM/ES, which supports functional stimulus patterns as well as acceptance and test quality criteria which relate to requirements and enable requirements coverage definitions. As such, our approach facilitates horizontal and vertical reuse by the unified notation of the enhanced CTM/ES. We present a concept for testbench execution automation to reduce cost- and time intensive manual human intervention in the verification process. The new notation is combined with representations and methods for the definition of a verification plan for constraint randomized tests derived from current approaches for automatic testbench generation in the domain of electronic system level design.

The remainder of this article is organized as follows. Our approach is presented in Section III. after the discussion of related work and a short introduction of basic principles of the test of combined HW/SW systems. Thereafter, we briefly sketch the implementation and close with a conclusion.

II. Related Work

The increasing demand for verification at earlier design steps led to the creation and introduction of methods and languages for *functional verification*. The industrial adoption of the methodology is supported through verification environments, which are implemented with standard *Hardware Verification Languages* like SystemVerilog, PSL, and *e*. Meanwhile, libraries and methodological guidelines have become available to supplement tooling and standardization efforts, such as the *Verification Methodology Manual for SystemVerilog* and the *Open Verification Methodology*.

On the other hand, existing methods for verification and testing of continuous control systems lack in expressivity and do not cover all areas of functional verification. E.g. test processes in the automotive industry are affected by tool-intensive and technologically heterogeneous test infrastructures [8]. In automotive systems development a product has to pass tests at several levels of abstraction such as Model-in-the-Loop (MIL), Software-in-the-Loop (SIL) and Hardware-in-the-Loop (HIL) tests and different tools are applied for this purpose like MTest and AutomationDesk [7].

Several exchange formats were introduced for the test of automotive systems. TestML [8] has been developed as an exchange format for functional, regression, and back-to-back tests and stimuli definitions can be captured by means of existing such as CTM/ES. The Classification Tree Method for Embedded Systems (CTM/ES) [6] is defined by use of set theory and the notion of mathematical functions. The latter are used to define interpolations between the distinct supporting points acquired by the Classification Tree Method. Additionally, there are efforts to standardize the Automatic Test Mark-up Language *ATML* [9] as an XML-based language.

Coverage for verification of combined HW/SW models, as modeled and simulated by MATLAB/Simulink, for instance, is limited to structural coverage of the model and/or the generated code [1]. An overview of coverage for continuous and mixed discrete-continuous systems is provided in [3]. Coverage criteria, which include data flow coverage, for instance, by means of equivalence classes are presented in [2]. A statistical coverage metric has recently been proposed for verification of continuous and mixed systems in [12]. The coverage of state regions with some timing information is described in [10].

III. Testing Combined Hardware and Software Systems

Verification of discrete systems applies functional verification based on object-oriented languages, while the mixed-signal domain mainly uses trivial stimuli and requires considerable manual effort for elaboration of results [4]. A methodical gap from requirements exists for several reasons: a methodical support for derivation of test descriptions requires a suitable target. As most requirements allow an infinite number of possible stimuli, a directed stimulus definition,

978-3-9810801-6-2/DATE10 © 2010 EDAA

i.e., a signal trace, cannot capture such a requirement, as it represents a single instance of stimulus only. However, the CTM/ES provides a first step for the definition of functional stimulus patterns, which can capture operational ranges of systems. The definition of requirements based acceptance criteria for automatic acceptance evaluation is usually only possible by definition of acceptance predicates. Existing predicate languages do not cover the definition of characteristic acceptance criteria for continuous systems. Existing proposals for test quality criteria are usually based on structural coverage criteria, which do not easily relate to requirements. Moreover, the reuse of test descriptions is only possible with high effort. Existing methods for functional verification and testing of HW/SW systems lack in expressivity and do not cover all areas of functional verification. Moreover, there is no accepted and standardized test definition language for combined HW/SW systems. Additionally, the original Classification Tree Method for Embedded Systems (CTM/ES) has pointed out first directions for the generic functional definition of tests for HW/SW systems [5]. The graphical notation of the CTM/ES is used for the definition of stimuli, which are interpreted and interpolated to a continuous waveform, before being discretized and feeds the model-under-test. Automatic acceptance evaluation is performed by correlation to a reference model, or by predicate evaluation [8]. Test quality is assessed by structural coverage. The stimuli and the acceptance model are derived manually from the requirements. The link of structural coverage to requirements is rather weak, as the type of coverage for the whole model can only be chosen from a limited set of coverage models [13].

IV. A Unified Method and Notation

The main goal of our approach to systematic testing of HW/SW systems is to narrow the methodical gap between requirements and testbenches. Figure 1 gives an overview of our concepts for systematic testing of HW/SW systems. The verification plan is based on the requirements. For increased flexibility and precision of stimulus specifications, constraints are applied for declarative stimulus definition including acceptance criteria and functional coverage based on the graphical notation of CTM/ES..

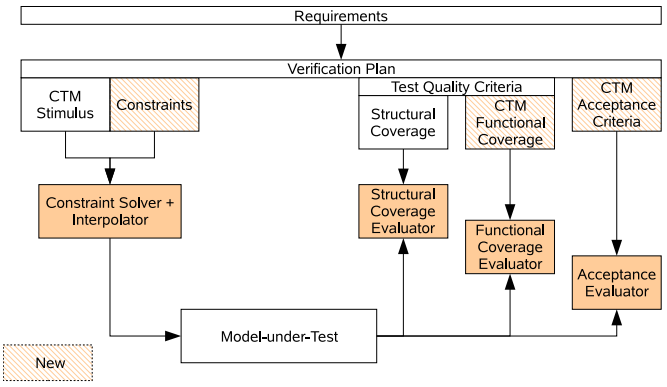

Fig. 1. New Unified Notation and Verification Plan for Systematic Testing

A. Stimulus Patterns

Stimulus definition with constraints allows requirement based stimulus pattern definitions, which can be more accurately targeted for improved test quality: The declarative nature of constraint-based stimulus patterns enables automatic generation of a wide range of stimuli. Additionally, as more implementation details become available, the declarative stimulus patterns can be adapted in a straightforward manner.

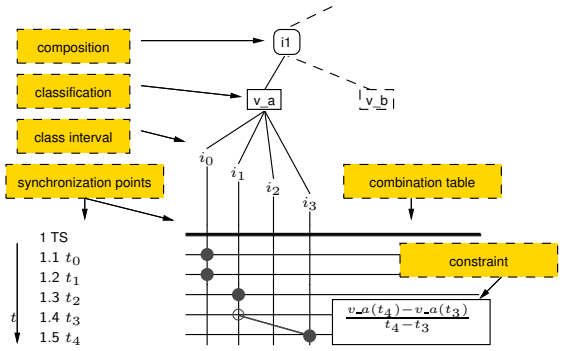

Fig. 2. Classification Tree with embedded Constraints

In CTM/ES, the classification tree is derived directly from the technical interface of the system under test, i.e., each input of the model-under-test is represented as a *classification* in the tree. Each input domain is partitioned into intervals or single values represented as *classes* below the accompanying classification. Figure 2 shows part of a verification plan classification tree derived from a module *M1* with an input interface *i1* with two inputs *v_a*, and *v_b*. The first composition node of the classification tree represents the interface *i1*. The next lower classification nodes represent an input interface. Here, the classifications define the signals v_a and v_b. The corresponding interface definition in SystemVerilog is:

```
interface i1;                  // from composition
  real v_a;                    // from classification
  real v_b;                    // from classification ...
```

The leaf nodes in the CTM/ES classification tree are denoted as classes, and they consist of non-overlapping value intervals which represent characteristic value ranges for the respective interface (cf. fig. 2). For the definition of test sequences in SystemVerilog, a data type for synchronization point definition is used, which captures the value *v*, the time *t*, and the interpolation type *ipol* for one signal:

```
class Int_sp;
  rand int v;
  rand time t;
  interpolationtype ipol;     ...
```

where *time* is a native SystemVerilog type for representation of time, and *interpolationtype* is an enumeration type, which corresponds to the different interpolation types supported by the CTM/ES. Both value *v*, and time *t* are declared randomizable variables. A corresponding class has to be defined for floating point values derived from generated integer values.

Constraints are not restricted to the value domain, rather than they may also be applied across synchronization points

affecting the timing conditions. This is important especially for the definition of dynamic constraints for stimulus patterns. They enable, e.g., definitions that control instantiated step heights as well as control of minimum and maximum steepness of ramp functions. The additional constraint is annotated to the test sequence of a classification tree as shown in figure 2 for the sequence TS. It shows a step interpolation after its second synchronization point and a ramp interpolation after its fourth synchronization point. The following constraint restricts the step height to a certain interval i.

```
v_a[2].v - v_a[1] inside {i};
```

The steepness of a function in general is constrained by $min(j) \leq \frac{\Delta v}{\Delta t} \leq max(j)$ with an interval j. The following term constrains the steepness of the ramp function of figure 2 to such an interval j.

```
(v_a[4].v-v_a[3].v)/(v_a[4].t-v_a[3].t)inside{j};
```

B. Acceptance Criteria

The new notation for acceptance criteria complements the stimulus pattern definition and it enables automatic acceptance criteria generation together with stimuli generation for fully automatic testbench execution. The notation provides for definition of tolerance bands by means of synchronisation points for the model-under-test response. These tolerance bands are dynamically instantiated in conjunction with the stimulus. The definition of a functional relation between stimulus- and acceptance synchronization points provides the reference model to the model-under-test. Instantiation data generated by constraint solving is applied to the acceptance evaluation set of synchronization points, which are interpolated to derive the tolerance borders for the model-under-test, before they are evaluated with the measured response of the model-under-test. The proposed solution describes an enhanced CTM/ES notation, which offers *a)* A definition of acceptance criteria synchronization points synchronized to stimulus pattern synchronization points, and *b)* a definition of functional CTM/ES classes to establish a functional relationship to the stimulus.

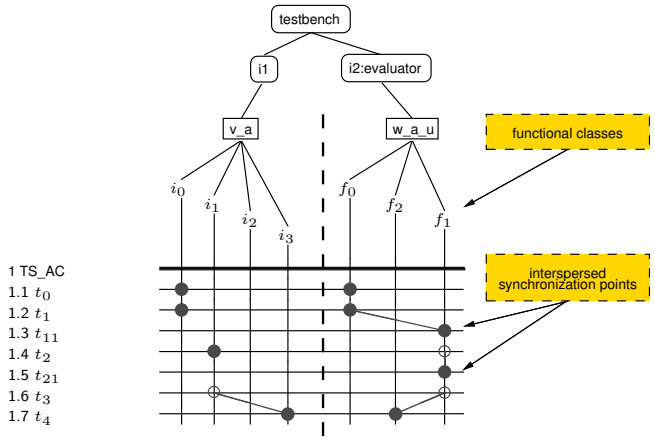

Fig. 3. Upper bound as acceptance criterium for *w_a*

In figure 3 an exemplary acceptance supremum *w_a_u* for a response signal *w_a* is shown in a classification tree together with its related stimulus *v_a* from abstract test sequence *TS* of figure 2. The classification tree represents a testbench definition with stimulus defined below the interface *i1* definition, and with acceptance definition defined below the interface *i2:evaluator* definition. The *evaluator* keyword marks this part of the tree as an acceptance criteria definition. The classification tree shows a classification node for the stimulus signal *v_a*. The classes shown for *v_a* are identical to the classes in the abstract test sequence definition in figure 2. Acceptance criteria below the *i2:evaluator* definition are formulated for a supremum *w_a_u* by means of three functional classes $f_{0,1,2}$. A functional class defines a function, which provides the value of its classification at a particular synchronization point. The function depends on the set of instantiated synchronization points and on the current synchronization point: $f(SP, csp)$, where SP is an ordered set of synchronization points, and csp is the current synchronization point. To maintain causality, only synchronization points of lower index (earlier) than csp should be considered for calculation.

C. Test Quality Criteria

Test quality criteria define verification goals. These criteria encompass structural coverage metrics, usually. Structural coverage metrics, however, do not enable the derivation of requirements coverage. Recently, additional test quality criteria have been introduced by means of functional coverage. Moreover, there are no approaches to functional coverage definition for HW/SW systems, which seamlessly fit into a verification process. A new approach to functional coverage definition for HW/SW systems is defined, which relates to requirements and enables requirements coverage derivation.

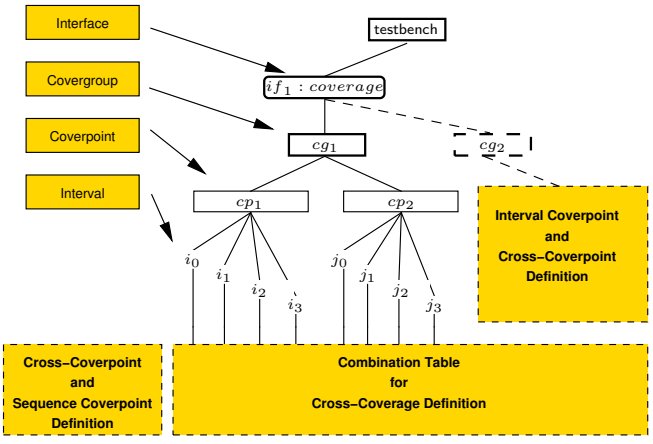

Fig. 4. Upper bound as acceptance criterium for *w_a*

An enhanced abstract classification tree for functional coverage is shown in figure 4. This classification tree presents the new areas of functional coverage definition within a classification tree. The root node corresponds to the testbench, and the next lower CTM combination nodes represent interfaces for coverage measurement, e.g., interface if_1. For each interface a number of covergroups can be defined as another set of CTM combination nodes ($cg_{1,2}$). Such covergroups encompass a set of coverpoints ($cp_{1,2}$) as CTM classifications, each

of which corresponds to a signal of the interface if_1. To the left of the combination table, a *cross coverpoint* can be declared for a set of lines in the combination table and appropriate logical markers placed in the combination table for the selection of relevant interval combinations. *Sequence coverpoints* define sequential, timed combinations of value intervals for a particular signal. They are defined in a similar manner as stimulus sequences, with time tags to the left of the combination table and the sequence defined by markers in the combination table. Cross coverpoints may also be defined across sequence coverpoints, such that concurrent sequences of several signals are covered. A notation similar to the previous cross coverpoints is used to define *cross sequence coverage* and to select from individual sequences defined in other coverpoints. The coverage calculation behavior can be modified by *option* annotations to bins, coverpoints, and covergroups.

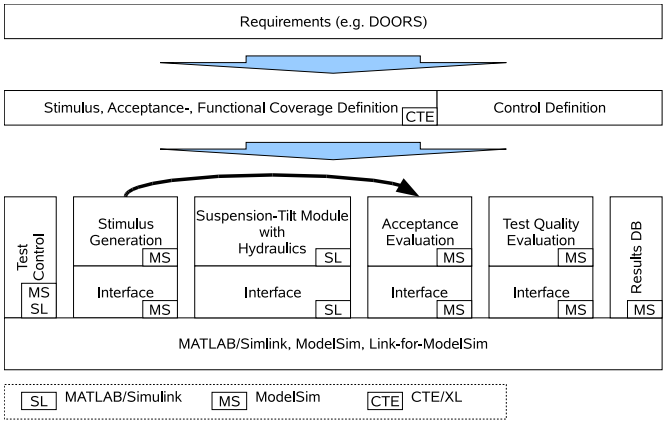

Fig. 5. Architecture and Tools

V. Testbench Implementation

Figure 5 depicts our implementation of the previously outlined method. Informal requirements are captured and organized by a requirements management tool, e.g., DOORS. Requirements are captured by textual or by visual means, before they are formalized and captured as features, stimulus patterns, acceptance criteria, and functional coverage criteria as well as control definitions. Patterns and criteria are captured by means of classification trees. A tool for classification tree definition is a *Classification Tree Editor*[15]. The test control is defined in the native language of the underlying execution environment. Here, we use SystemVerilog and MATLAB/Simulink syntax. From these definitions the testbench can be created automatically for the execution environment, which consists of MATLAB/Simulink [14] and Modelsim [11]. The combined HW/SW system model is defined and executed in MATLAB/Simulink. Most other components, such as Stimulus Generation, Acceptance Evaluation, Test Quality Evaluation are generated in SystemVerilog and executed in Modelsim. For this purpose a set of tools is applied, which generate SystemVerilog code in combination with Modelsim interfaces from classification trees. The results database is a feature of Modelsim. Parts of the control description is described and executed in MATLAB/Simulink and in Modelsim with the Link-for-Modelsim.

VI. Evaluation and Conclusion

We have performed an experiment with 4 classification tree test sequences on a suspension-tilt module with a 3 dimensional movement freedom, which demonstrated the efficiency of our approach. The example achieved a structural coverage of 83% in Simulink. Inspection of the structural coverage report and of the model revealed that the only missing coverage was on a set of integrators with result limits, deeply embedded within the model, and difficult to control. Functional coverage, however, revealed that testing of 2 of the 3 axes of module movement was incomplete with respect to the requirements. The definition of a unified CTM/ES notation for stimulus, acceptance criteria, and test quality criteria enabled exchange and reuse of information between, e.g., the stimulus and functional coverage aspects of the notation. Functional coverage definition could be assisted by existing stimuli as a pattern for coverage model definition and for identification of requirements.

Acknowledgments

This work was partly supported by the DFG Sonderforschungsbereich 614 and by the BMBF through the ITEA2 project TIMMO (01IS07002).

References

[1] W. Aldrich, "Using Model Coverage Analysis to Improve the Controls Development Process," in *AIAA Modeling and Simulation Conference*, Monterey, California, Nov. 2002.
[2] V. Alyokhin, B. Elbel, M. Rothfelder, and A. Pretschner, "Coverage metrics for continuous function charts," in *Proceedings 15th IEEE Intl. Symp. on Software Reliability Engineering*. St. Malo, France: IEEE, Nov. 2004.
[3] A. Baresel, M. Conrad, S. Sadeghipour, and J. Wegener, "The interplay between model coverage and code coverage," in *Proceedings of EuroCAST*, 2003.
[4] H. B. Carter and S. G. Hemmady, *Metric Driven Design Verification*. Springer, 2007.
[5] M. Conrad, *Modell-basierter Test eingebetteter Software im Automobil (Model-based Testing of Embedded Automotive Software)*, ser. PhD Thesis. Wiesbaden: Deutscher Universitäts-Verlag, 2004.
[6] M. Conrad, "A Systematic Approach to Testing Automotive Control Software," in *Proc. 30. Int. Congress on Transportation Electronics (Convergence '04)*, Detroit, MI, USA, Oct. 2004.
[7] dSPACE, Homepage. www.dspace.de
[8] J. Grossmann, M. Conrad, I. Fey, A. Krupp, K. Lamberg, and C. Wewetzer, "TestML – A Test Exchange Language for Model-based Testing of Embedded Software," in *Automotive Software Workshop '06*, San Diego, Mar. 2006.
[9] IEEE SCC20 ATML Group, "IEEE ATML specification drafts and IEEE ATML status reports," 2008. grouper.ieee.org/groups/scc20/tii/
[10] A. A. Julius, G. E. Fainekos, M. Anand, I. Lee, and G. J. Pappas, "Robust test generation and coverage for hybrid systems," in *Hybrid Systems: Computation and Control*, ser. LNCS 4416, 2007.
[11] Mentor Graphics, Homepage. www.model.com/
[12] T. Nahhal and T. Dang, "Test coverage for continuous and hybrid systems," in *CAV 2007*, ser. LNCS 4590, W. Damm and H. Hermanns, Eds. Berlin: Springer, 2007.
[13] J. Schäuffele and T. Zurawka, *Automotive Software Engineering*, 3rd ed. Wiesbaden: Vieweg, Mar. 2006.
[14] The Mathworks, Homepage. www.mathworks.com
[15] J. Wegener and R. Pitschinetz, "Classification-Tree Editor CTE/XL ," 2008. www.systematic-testing.com