Intelligent Test Vector Generation for the Functional Verification of Hardware Designs from Temporal Logic Assertions

Ralph A. Marczynski

Abstract—Due to the rapid growth of functional complexity in hardware designs and the limited progress in functional verification with respect other design stages, verifying design correctness has become a bottleneck in the modern digital system design cycle. In an effort to improve current ad hoc simulation processes, this paper purposes a method for the automated generation of simulation vectors based on user-defined assertions. The proposed development efforts focus on an algorithm for reusing temporal logic assertion prepared for formal verification techniques to generate vectors that can “prove” the properties through simulation, a useful method when the size of a design exceeds a formal verification tool’s limitations. Distinctions among different temporal logics are discussed and candidates for this method are identified. Model Checking and other (semi-)formal methods are overviewed and portable approaches are discussed.

Index Terms— Formal Methods, Functional Verification, Model Checking, Simulation, Temporal Logic.

I. INTRODUCTION

Due to numerous advances in the production of digital systems and growing consumer expectations, design sizes continue to grow exponentially with Moore’s Law. Such progress in the semiconductor industry continuously moves us towards one billion transistors on a single chip. This can be accredited to progress in both the physical Integrated Circuit (IC) layer as well as advances in the ever-important tools used for design and implementation. Historically, tool development has been focused on important issues such as logic-synthesis and place-and-route [1]; unfortunately, major practical advances in functional verification have come few and far in between. This gap in Electronics Design Automation (EDA) tool development puts our ability to effectively utilize the number of available transistors at risk.

Functional verification is the process of determining some level of confidence that a design meets its specifications. (As opposed to Testing, which verifies the implementation). The current trend is the verification of designs through simulation [1], consisting of sending input to the design under test and observing the output. Typically, the design is judge by the equivalence found when comparing the output with that of a reference model exposed to the same stimulus (usually the design at a higher level of abstraction). This approach is by far the most common verification method in the production environment; unfortunately this in not due to the method’s effectiveness, but rather the lack of superior options.

The shortcoming of simulation verification is the limited functional coverage. Even for modest designs, the number of vectors needed to for an exhaustive test is too large to simulate in a feasible amount of time. In order to utilize simulation, one needs to choose the test vectors carefully as to gain the maximum amount of coverage and ensure tests for rare “corner-cases”. Automated methods for such intelligent vector generation are the saving grace of simulation, but this area still requires further study to reach its full potential.

Formal Methods provide an alternative capable of overcoming the obstacles faced in simulation. Formal Methods is the application of mathematical methodologies to the specification and verification of systems [2]. The key advantage is the ability to exhaustively test the design with respect to its specification. Encapsulating several methods, the technique of Model Checking [3] is emerging as the most practical Formal Method, with commercial tools such as [4][5] employing the techniques. Unfortunately, even “industry-strength” implementations are limited in the size of the design they can handle; meaning manual portioning and distribution among different verification methods is necessary.

With many mature to relatively mature technologies plagued with obstacles and the need for burdensome human intervention; a workable, practical, and effective verification flow appears to be rooted in the integration of multiple verification methods [6][7][8]. Specifically, in [6] and [7], Dill makes several calls for the further development of semi-formal methods focused on automated test vector generation, as well as limiting the power of Model Checking in order to “find bugs” rather than exhaustively prove a system’s correctness.

This paper proposes the investigation of methods utilizing such integration and reuse. Due to Model Checking’s industry acceptance, the technique will serve as the core of all diverging methods. When Model Checking is overcome by the size of the state space, existing temporal logic assertions found in the specification will be reused to generate the test
vectors for simulation verification. Integrating additional formal and semiformal methods will be necessary for such vector generation. Clearly, (due to the failure of Model Checking) complete state space enumeration (whether symbolic or explicit) cannot be achieved. Other approaches must be therefore examined to see if they encapsulate any portable approaches, either directly or with modifications. Current candidates are decompositional model checking [9][10][11][12][13], bounded model checking [14][15][16], symbolic simulation [17], and symbolic trajectory evaluation [2][18].

The remainder of this paper is organized as follows. Section 2 presents some general background on temporal logic and model checking. Section 3 outlines the steps necessary for the evaluation of the proposed method with respect to related work. Section 4 outlines the research plan in terms of a timeline. Finally, section 5 concludes the paper with a brief discussion of expected results and potential extensions.

II. BACKGROUND

A. Temporal Logic

Temporal Logic is a logic (usually prepositional or first order logic) augmented with temporal modal operators which allow reasoning about how the truth values of assertions change over time [19]. This quality allows the use of temporal logic to formulate specifications (properties) about the behavior of a design. The truth of a formula in temporal logic is understood relative to a system $M$ which is perceived to be in one of a (possibly infinite) set of states $S$ at any point in time, and which performs transitions between states as time progresses. Atomic prepositions are associated with each state in $S$, forming the basic building blocks for temporal formulas [2].

Different types of temporal logics exist, classified in-depth in [19]. This paper only distinguishes between a subset of all the possibilities. As the most drastic difference, temporal logic can be classified as linear or branching. In linear temporal logic (LTL) we are concerned with a specific path, consisting of only one future, where as branching temporal logic (BTL) can have any number of possible futures.

The main difference between linear- and branching- time logic is the lack of existential quantification over paths in LTL [19]. The expressive power of LTL is limited to that of BTL variations, but there exist different schools of thought as to the limitations a chosen logic may have when reasoning about concurrent systems [2][20]. In terms of model checking, Computation Tree Logic (CTL, a type of BTL), is more commonly used due to a simpler verification algorithm. This stops being the case, however, when one considers open-systems (systems where non-determinism comes from external source, i.e. input) [20]. The extension of CTL to open systems with inputs is achieved by augmenting the state space with a component representing the current input, and extending the transition relation to allow non-deterministic changes to this component [9]. Supporting the right form of temporal logic (CTL, BTL, or a specific variant of the two) appears to be crucial to the success of this proposed method, as it will directly involve computation complexity, property modeling, and usability due to varying levels of expressiveness.

B. Model Checking

Model checking is the process of verifying that a given design satisfies a set of temporal formulas [2]. (A more general definition of model checking is presented in [19].)

To perform such a procedure, the finite structure $M$ must undergo state space enumeration, performed either explicitly or symbolically. A model checking algorithm for CTL using explicit state modeling was first introduced by Clarke et al. in 1986 [21]. This required the construction of the entire state transition graph that is a least linear in the size of the model’s reachable state space, which in turn is often exponential in the number of state holding elements [2]. Such resource requirements rendered this technique impractical for even very modest designs.

In 1987, McMillian realized that by using a symbolic representation utilizing BDDs, much larger designs could be verified [3]. Although this made model checking more practical, memory requirements were still too demanding to apply the procedure to common complex systems.

Approaches have been investigated in reducing the BDD representation even further through “exotic” BDD forms as well as design abstraction. This resulted in commercial implementations of the technique [4][5], but the approach is far from being a complete solution.

In the following, an approach is proposed that aims at utilizing the present temporal logic assertions used in model checking in order to create a “smooth” and effective transition to simulation based verification once model checking proves infeasible.

III. APPROACH WITH RESPECT TO RELATED WORK

Techniques have been developed using advances in formal methods to generate “intelligent” vectors for increasing functional coverage. In [22], Ho defines a procedure for generating a control interaction graph (incorporating the design’s FSM and relevant control logic) that undergoes a traversal to create vectors for initiating as many control interactions a possible. In [23], a similar approach is presented based on a different metrics for the identification of such vectors. However, both approaches depend on a state space enumeration (explicit in [22] and symbolic in [23]) and are therefore subject to the same limitation as model checking.

The technique proposed in this paper strives to eliminate the need for state space enumeration, whether symbolic or explicit. Using temporal logic assertions prepared for model checking as guide, the technique attempts to create test vectors by borrowing from approaches developed for model checking without state space enumeration.

Decompositional Model Checking techniques not requiring a complete state space representation, although impractical for model checking, seem to be the most promising direction towards such a solutions. In [9], a technique is defined using a preorder of structures that captures the relation between a
component and a system containing the component. Using assume-guarantee-style reasoning, satisfaction of a temporal formula corresponds to being below a particular structure in the preorder.

In [10], an algorithm is given for checking whether a system satisfies a specification is all environments. [11] uses module specifications consisting of guaranteed behavior and assumed behavior to perform modular model checking. [12] offers an algorithm for generating necessary model checking components in an on-the-fly manner. And finally, [13] offers a technique that exploits the compositional structure of an introduced temporal specification.


The above techniques will have to be explored to identify promising methods, their adaptability to various specification logics, and their overall compatibility with a useable framework (expressiveness, algorithmic complexity, etc.)

Bounded Model Checking (BMC) additionally seems applicable. BMC [14][15][16] considers counterexamples of a particular length $k$ and generates a prepositional formula that is satisfiable iff such a counterexample exists. Such “partial” model checking techniques, considering LTL specification, do not require complete state space enumeration and definitely deserve further investigation.

Through the course of evaluating such approaches, simulation methods beyond the traditional form will be considered. Symbolic Trajectory Evaluation [2][18] and Symbolic Simulation [17] offer much more coverage during each iteration in a simulation procedure. To the best of my knowledge, no efforts have been initiated to choose “intelligent” symbolic stimulus.

Upon an evaluation of existing procedures, deeper understanding of temporal logic, it’s capabilities, modeling, and manipulation must be understood. Specifically, previously implemented approaches must be explored to gain insight as to property representation and meaningful interpretation.

Current progress calls for a deeper understanding of the theoretical aspects of modeling open and closed systems. With non-determinism introduced via the input, approaches have to be investigating to model and append the non-deterministic environment to closed systems. Since we are concerned with generating discrete test vectors, it is currently believed that the system can remain closed with a method of iteratively expanding the non-deterministic environment. A feasible modeling technique will surely influence the temporal logic supported, hopefully by allowing for a superset of existing logics rather than a subset.

IV. TIME-LINE

Further investigation will be broken down as follows.

April 12, 2003 – Sufficient knowledge of Model Checking algorithms (including decompositional approaches) is expected, in terms of implementation and theoretical background.

April 19, 2003 – Temporal Logic will be understood at a greater level, specifically in terms of the non-deterministic environment (input) modeling in closed systems as it applies to different variations of temporal logic (see [19]).

April 26, 2003 – A theoretical understanding of constructively building a deterministic model (encapsulating the test vectors) representing the non-deterministic input environment and how it can be merged with an existing closed system model $M$ will be achieved.

May 2, 2003 – The procedure will have been considered in terms of real concurrent systems defined in an HDL: the necessary FSM and control logic extraction (reusable tools for this task will have been identified), system modeling, implementable representation, and a traversal algorithm for generating the simulation vectors.

The expected results at the close of the Spring 2003 semester will be a high-level understanding and foundations of the algorithm, in terms of procedures and data structures.

V. CONCLUSION

The Spring 2003 semester results will provide a high level understanding of which temporal logic (see [19]) can be exploited, how it can be represented, and how it can be interpreted to extract intelligent simulation vectors. In essence, the result will shed light on how simulation can be used to “challenge” temporal logic specification. The theoretical background for non-deterministic environment merging with a closed system will be presented, in terms of its construction and influence on the system model (should it remain closed, should the environment be symbolic in nature, etc.).

Multiple extensions of these limited, yet important, results will be possible. The primary extension can consist of a tool realization, feasible in terms of usability and complexity. Secondly, these development efforts can provide some theoretical understanding as whether or not proving temporal logic assertions (an in turn system correctness) can be achieved through simulation using only a subset of all possible test vectors.

As the final noted extension, the procedure can be possibly extended to not only generate intelligent input, but additionally the corresponding output. Such a method would do away for the need of reference models, as they are currently understood. In essence, this extension can be an attempt at using the temporal logic assertions as the reference model.

REFERENCES