The Maximal VHDL Subset with a Cycle-Level Abstraction

Wendell C. Baker and A. Richard Newton
{wbaker,newton}@eecs.berkeley.edu
Department of Electrical Engineering and Computer Science,
University of California, Berkeley, CA 94720, USA

Abstract

The maximal VHDL subset with a cycle-level abstraction is defined. This subset requires that the description have three semantic properties: responsiveness, modularity *and* causality, *but full VHDL is neither modular nor causal.* Synchronous VHDL *is the responsive, modular and causal subset of VHDL. The compiler uses* modularity-checking *and* causality-checking *to identify admissible programs*.

1 Introduction

The extraction of a finite state machine from an HDL-based description has become an important aspect of the current generation of language-based design methodologies. Such techniques have application in numerous areas: in formal verification when establishing the behavioral equivalence of two system descriptions (*c.f.* [5] [17]); in synthesis when the implementations in hardware or software must be faithful to the behavior of the language-based specification (*c.f.* [4] [9]); and in high-performance simulation when cycle-level approximations are used (*c.f.* [22]).

The current generation of HDLs are based on the discrete-event paradigm in which components are activated under the control of an event-processing loop. This provides for a very flexible and general operational modeling capability. At the same time however, it has precluded the use of HDL programs as abstract behavioral specifications. To date, there has been no theory which explains how behaviors defined by discrete-event models relate to the computational model of hardware: finite state machines. Existing approaches have applied *ad hoc* style guidelines in an effort to constrain the problem for synthesis or verification (*c.f.* [5] [8] [10] [19] [20]). Such proposals have met with mixed success since all are different and all are justified by the idiosyncratic needs of a particular user community or design tool. What is needed is a *general theory* which addresses the behavioral extraction problem at the semantic level and thereby subsumes any syntax-based policy guidelines.

The microsemantic theory described in this paper fills this need. It precisely explains the relationship between discrete-event and finite state machine semantics and with it, the finite state machine extraction problem can be stated in fully general form. Using this theory, the Synchronous VHDL subset is defined as the largest general subset of VHDL with a cycle-level abstraction.

The observations in this paper build upon recently-reported results in the analysis of discrete-event semantics [14]. The additional new result reported here is a domain-theoretic construction of a fully abstract semantics for VHDL within its non-abstract semantics of δ-time. The result presented in this paper holds for any system description language where time has a fine structure (the so called “δ-time” or “micro”-time) and a zero-delay assumption.[1] Such languages include HDLs based on discrete-event simulation: VHDL [15] [16] and Verilog [27]; as well as synchronous languages [13] such as Esterel [4]. For concreteness, and because of its widespread use, we report on VHDL-1076-1987 exclusively in this paper.

2 Semantics for State-Transition Systems

Our approach draws heavily from results in formal model theory and language semantics [12] [25]. This section presents the basic definitions.

2.1 The Semantic Map

A semantics $\mathcal{S}$ is an abstract map from elements of a language $\mathcal{L}$, that is program instances, to elements of a mathematical model $\mathcal{M}$. This is written:

$$\mathcal{S}[\![\, program \,]\!] = model$$

A semantics is *denotational* when association at the syntactic

1. Also referred to as the *perfect synchrony hypothesis* [4].

EURO-DAC ’96 with EURO-VHDL ’96
0-89791-848-7/96 $4.00 ©1996 IEEE

level is defined in terms of composition at the model level. That is, $\mathcal{S}$ is defined in terms of one or more composition operators $\circ : \mathcal{M} \times \mathcal{M} \rightarrow \mathcal{M}$. For example $\mathcal{S}$ might contain mappings rules of the form:

$$\mathcal{S}[\![\ statement_1 \texttt{;} statement_2\ ]\!] = model_1 \circ model_2$$

$$\mathcal{S}[\![\ statement_1 \texttt{||} statement_2\ ]\!] = model_1 \times model_2$$

In turn, the $model_i$ are computed by applying $\mathcal{S}$ to the structural sub-parts of the original element of $\mathcal{L}$. As indicated, the composition operators may vary with the kind of statement mapped by $\mathcal{S}$.

The models are defined in terms of Scott's domain theory where a *domain* is a countable set endowed with an ordering relation "$\sqsubseteq$" which is understood to mean "approximates." A domain has a minimal element $\perp$, called "bottom" which is the unique representation of "no information." This framework allows for a precisely-defined notion of approximation and limit points. In turn this allows for the definition of a (large) computation as a finite series small steps; the analogy between the cycle-level and δ-steps in VHDL is direct.

A semantics is said to be *fully abstract* when it does not preserve any structure from the source language [23]. A semantics is *existential* when no approximation is involved: $\mathcal{S}$ identifies model elements directly and explicitly. In contrast, a semantics is said to be *computational* when $\mathcal{S}$ identifies model elements by a series of approximations tending to a limit. This notion of approximation and limit points is fundamental to the development which follows.

2.2 A Semantically-Defined VHDL Subset

Our goal is the weakest set of constraints on semantic domains and δ-steps, which are applicable *a priori*, such that a VHDL program is guaranteed to have a cycle-level abstraction.

2.3 Finite-State Models

The model of computation underlying cycle-level behavior is the finite-state transition system.

2.3.1 The Fully Abstract Model $\mathcal{M}$

The fully abstract FSM model is defined in the traditional manner [18] save that the states, inputs and outputs are defined on finite domains instead of merely finite sets.

$\mathcal{M} = (Q, I, O, T, Q_0)$
where:
Q is a finite set of states; Q is a flat finite domain.
$I = (i_0, i_1, \ldots, i_m)$ is a vector of input signals i_j defined over flat finite domains; the domain of I is the Cartesian product of the domains of the i_j; and the alphabet Σ_I is the observable subset of these vectors.
$O = (o_0, o_1, \ldots, o_n)$ is a vector of output signals; the signals o_j, the domain of O and the alphabet Σ_O are defined similar to I.
$T \subseteq Q \times I \times O \times Q$ is a transition relation governing the transitions of the machine.
$Q_0 \subseteq Q$ is a set of initial states.

The elements of the model $\mathcal{M}$ are the steps among the states Q which are allowed by T, restricted to Σ_I and Σ_O.

The model $\mathcal{M}$ is fully abstract because it does not represent any information about what occurs within a step. At the fully abstract level, all concurrent coordination which might have been intuitively visible in $prog \in \mathcal{L}$ has been "compiled away."

2.3.2 The Non-Abstract Model $\mathcal{M}_\delta$

Such simplicity is not the case in the non-abstract model because transitions between macro states (*e.g.* $q_1 \rightarrow q'_1$) are not atomic. Rather, they are defined in terms of compound δ-*transition paths*. The non-abstract model is:

$\mathcal{M}_\delta = (Q, T_\delta, Q_0)$
where:
$Q = S \times I \times O$ is a Cartesian product domain of the state and the domains:
$S = S_T \cup S_\delta$ where S_T is a flat domain of macro-states; S_δ is the non-flat domain of δ-states.
I and O are Cartesian product the domains just as with the fully abstract case.
$T_\delta \subseteq Q \times Q$ is a transition relation.
$Q_0 \subseteq Q$ is a set of initial states.

The ascending chains in S_δ are properly referred to as *control-flow paths in δ-time* and S is constructed so that: $\forall s_T \in S_T . \forall s_\delta \in S_\delta . s_\delta \sqsubseteq s_T$. Thus when a control flow path of δ-states reaches a macro-state it is "complete." At such a state T_δ is assumed to contain a self loop called an *absorbing end condition* which prevents further increase.

By convention, the initial states Q_0 are all macro-states (*i.e.* the $s \in S$ are from S_T not S_δ), the coordinates of I are not $\perp$ and the coordinates of O are $\perp$.

2.4 Problem Statement

The non-abstract semantics, $\mathcal{S}_\delta$, is computational whereas the fully abstract semantics, $\mathcal{S}$, was existential. Absent a particular subject language, the concern here is with the conditions which must exist in a non-abstract semantic model to allow a fully abstract model to be embedded in it through a *dimensional projection* [23] that suppresses the intra-step implementation details. This situation is depicted in Figure 1.

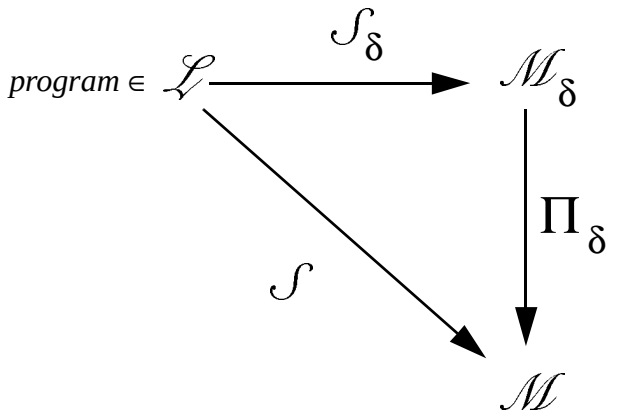

Figure 1. A non-abstract semantics $\mathcal{S}_\delta$, model $\mathcal{M}_\delta$ and a fully abstract semantics $\mathcal{S}$, model $\mathcal{M}$

The conditions of interest here are those which must hold in $\mathcal{S}_\delta$ and $\mathcal{M}_\delta$ so that the diagram of Figure 1 commutes. That is, for the admissible $program \in \mathcal{L}$, the following holds:

$$\mathcal{S}[\![\, program \,]\!] = \mathcal{S}_\delta[\![\, program \,]\!] \circ \Pi_\delta \qquad \text{(Eq 1)}$$

Here Π_δ is a dimensional projection which suppresses the orthogonal dimensions of $\mathcal{M}_\delta$ that contain implementation details.

The application of Π_δ to an element of $\mathcal{M}_\delta$ identifies the fully abstract submodel within $\mathcal{M}_\delta$ as:

$$\Pi_\delta((S_T \cup S_\delta) \times I \times O, T_\delta, Q_0) = (S_T, I, O, \hat{T}_\delta, \hat{Q}_0) \qquad \text{(Eq 2)}$$

where $\hat{T}_\delta$ is a projection of the infinite paths in the transitive graph of T_δ onto $S_T \times I \times O \times S_T$ and $\hat{Q}_0$ is a projection of Q_0 down onto S_T. Details can be found in Baker [3].

3 Limits on Microsemantics

There are three desirable properties of a non-abstract semantics: *responsiveness*, *modularity* and *causality.* There is also a theorem, the RMC Barrier Theorem,[1] that states that these three properties cannot appear together in a semantic model having the dimensional structure of $\mathcal{M}_\delta$. This result predicts the difficulty of embedding $\mathcal{M}$ as a submodel within $\mathcal{M}_\delta$ and by extension the difficulty in identifying a subset of VHDL semantics which is fully abstract.

3.1 Three Desirable Properties

Responsiveness (R or $\overline{R}$): A system is considered responsive if its output occurs in the same step as the input that caused it. A semantics $\mathcal{S}_\delta$ is responsive if it is possible to define a responsive system under the rules of the semantic map $\mathcal{S}_\delta$.

Responsiveness distinguishes between Moore and Mealy machines. The former is $\overline{R}$; the latter is R.

Modularity (M or $\overline{M}$): A semantics $\mathcal{S}_\delta$ is modular when environment-to-component and component-to-component communication is treated symmetrically.

Modularity distinguishes the broadcast model of inter-process communication. An equivalent statement is that signals have flat domains. A $\overline{M}$ semantics allows a test for the order in which outputs are produced within a step. A $\overline{M}$ semantics allows outputs to be assigned multiple times within a step. In both these cases signals cannot have flat domains.

Causality (C or $\overline{C}$): A semantics is causal if the system response does not anticipate its own future within a step.

This is the classical definition of causality from systems theory (*c.f.* [28]), but applied here solely within a step. Causality requires that there exist a partial order among coordinating components which is respected by concurrent composition. The partial order may be state-dependent for a given component.

With these definitions, a semantics $\mathcal{S}_\delta$ can be crudely characterized by the properties it possesses or lacks (*e.g.* $\overline{R}MC$, $R\overline{M}C$ or $RM\overline{C}$). Of the three properties:

- R is desirable for the cogency it affords (the Mealy- versus Moore-machine);
- M is desirable for the limitation that it imposes on the complexity of concurrent coordination (single assignment); and
- C is necessary when treating systems operating the a physical world where time moves forward (the future cannot be anticipated).

M is also crucial to the embedding of the full abstract model

1. Due to Huizing and Gerth [14].

$\mathcal{M}$ within $\mathcal{M}_\delta$. The fully abstract model $\mathcal{M}$ supports single assignment of outputs only therefore so must $\mathcal{M}_\delta$.

3.2 The RMC Barrier

Unfortunately, there is no *RMC* semantic map (details can be found in Huizing and Gerth [14] or Baker [3]). But *RMC* is *required* for the diagram of Figure 1 to commute. By definition $\mathcal{M}$ is *R* and *M*: Mealy machines can be represented and outputs are singly assigned in a step. $\mathcal{M}_\delta$ induces a (partial) relating inputs to outputs by its multi-step paths.

3.3 Surpassing the RMC Barrier

Fortuitously, the RMC Barrier applies to the class of semantic maps not to systems. This leaves open the possibility of having an incomplete or inconsistent semantics and admitting only program instances where *RMC* holds. In the case of synchronous languages [4] such as Esterel, the raw semantics is $RM\overline{C}$ and a subsequent *causality checking* step establishes *C* for the admissible programs [13].

4 Semantics of Discrete-Event Languages

Languages based on discrete-event semantics (DES) are studied based on the domain definitions of Section 2.3 and the RMC Barrier Theorem.

4.1 The Simulation Cycle

Discrete-event semantics is defined by an event processing loop which executes portions of the program by propagating representations of events. The event loop for VHDL (from [15], §12.6.3). An examination of the state transition behavior of the simulation cycle shows that it induces a *three-level* structure of time as depicted in Figure 2. This is somewhat at variance with the standard presentation of VHDL's "δ-time" in which a two-level structure is supposed. The explanation is that the invocation of an individual process is a step in the model. This gives a fine structure within δ-time which is here called "η-time" (equivalently "nano"-time).

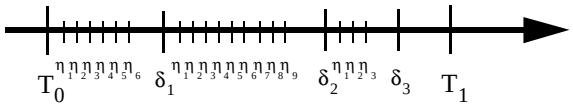

Figure 2. The three-level structure of time

4.2 Relationship to RMC Barrier

The three-level structure of time in discrete-event semantics is $R\overline{M}\overline{C}$ at the macro-time level and $\bar{R}_\delta M_\delta C_\delta$ at the δ-time level.

Theorem 1: DES is $R\overline{M}\overline{C}$.

Proof: (sketch, details in Baker [3])

Observing the LRM [15] and the simulation cycle, §12.6.3, one can see that:

Case R: Mealy-machines can be described.

Case $\overline{M}$: There exist VHDL programs where a signal takes on more than one value in a macrostep.

Case $\overline{C}$: There exist VHDL programs in which simulation time never progresses. That program has finite state so the infinite δ-steps are oscillatory; a cycle precludes the existence of a partial order. QED.

Theorem 2: DES is $\bar{R}_\delta M_\delta C_\delta$

Proof: (sketch, details in Baker [3])

Observing the LRM [15] and the simulation cycle, §12.6.3, one can see that:

Case $\bar{R}_\delta$: A signal assignment does not become visible until the next δ-step.

Case M_δ: A signal has a single value in a δ-step.

Case C_δ: A δ-step has a number of η-steps. A process is run at most once within a δ-step. This bounds the number of η-steps. The order the runnable processes occur is immaterial; any partial order suffices. QED.

5 Synchronous VHDL

With this background the Synchronous VHDL subset can now be defined. This new definition expands upon the previous definition [1] [2] with new semantic conditions based on microsemantic analysis and the RMC Barrier.

5.1 Syntactic Requirements

As before, and consistent with other cycle-level VHDL subset [19] [5], the admissible VHDL programs must have finite state. This implies proscriptions on the manipulation of time in waveform assignments via the **`after`** clause and aggregate waveform assignments themselves. Signal assignments must use only δ-delay. Also proscribed are the use of dynamic memory and reference to the external environment. These include the dependence upon a stack via static scoping, heap memory allocation via **`new`** and the **`access`** type constructor and the external environment via the **`file`** type.

5.2 Semantic Requirements

When a VHDL program's δ-time reaction terminates in an instant then there exists a domain construction and approxi-

mation relation "$\sqsubseteq$" such which *existentially* associates the η-transitions of the three-level operational semantics with the transitions of a two-level δ-time semantics [3]. This is depicted in Figure 3. The association is existential because the domain and its $\sqsubseteq$ relation need not be explicitly stated; it exists, *ex post*, if and only if every δ-step series is finite.

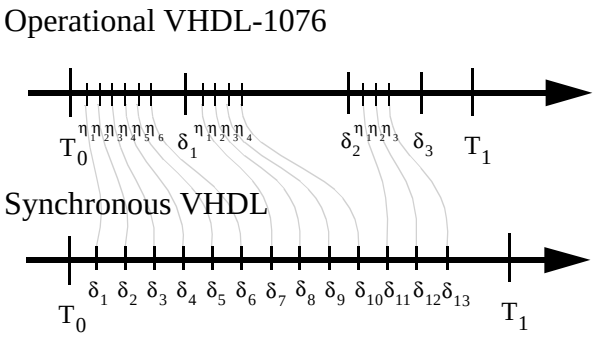

Figure 3. The Correspondence Between Operational VHDL-1076 and Synchronous VHDL

5.3 Requirements for Full Abstraction

The domains of $\mathscr{M}_\delta$ require that *RMC* hold on admissible programs. The analysis of Section 4 slowed that the unrestricted semantics of VHDL is $R\overline{M}\overline{C}$. So a VHDL program instance has a cycle-level abstraction just when it is *M* and *C*. This establishes the *modularity-checking* and *causality-checking* ("*MC*"-checking) problems.

Modularity-checking is a decision procedure that certifies that the signal usage in the design is consistent with flat domains. The modularity check ensures that every signal is assigned at most once on every δ-step path between macro states.

Causality checking is fundamentally a monotonicity check. Malik has shown that such a check for functions on the flat Boolean domain is NP-complete [21]. This recent result explains why early causality checkers (*e.g.* as reported for the early versions of Esterel [4]) only attempted a conservative estimate of monotonicity. Current work focuses on applying BDD-based symbolic representations [6] to the causality-checking problem (*c.f.* [24]).

5.4 A Prototype Implementation

We have implemented a prototype Synchronous VHDL compiler. Its overall architecture is fairly typical and is depicted in Figure 4. What is unique in the flow described there is the modularity- and causality-checking phase. Our original prototype used the Esterel V3 [7] compiler for this purpose though we have since formulated these checks directly on a language-independent abstract-instruction representation [3].

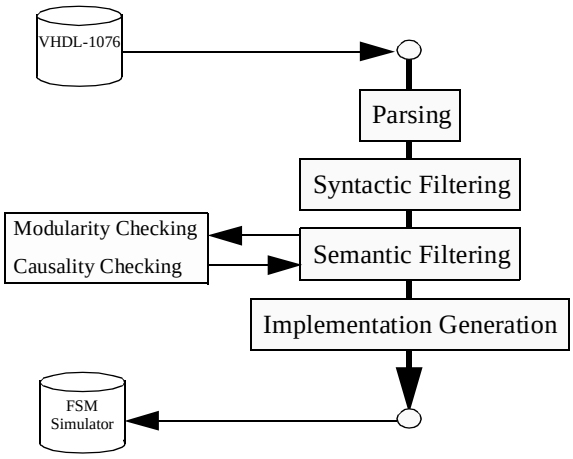

Figure 4. Architecture of a Synchronous VHDL Compiler

5.5 Experience

Our experience with this approach and with the Synchronous VHDL subset in particular has been mixed for a number of reasons. First, the *MC*-checking is a limiting factor. The MC-check is known to be a difficult problem in its own right though the use of clever symbolic representations [24] may help to alleviate this. Secondly, it is extremely cumbersome if not impossible to describe interrupt-like and nested sequential/concurrent behaviors in VHDL. This is a fundamental limitation of VHDL's flat process model. The only alternative here is the addition of these concepts at the language level The SpecCharts [11] is one such proposal, though there are severe problems with the naive use of that approach [26].

Most vexing of all is VHDL's $\overline{R}_\delta M_\delta C_\delta$ property which often causes signals to be multiply assigned. These are properly referred to as "glitches." An example is sketched in Figure 5. That example is not an admissible Synchronous VHDL design because ***O*** may be assigned twice in an instant for a change in ***I*** which also triggers an event on ***Q***. The way to alleviate this problem is to develop techniques for handling $\overline{M}$ semantics where signals have non-flat domains and support multiple assignment.

6 Conclusion

The extraction of a cycle-level FSM from an HDL system description is an important aspect of language-based design. The presentation in this paper defines the conditions under

```
entity EX is
   port(I: in IN_TYPE; O: OUT_TYPE);
end;
architecture Mealy of EX is
   signal Q: STATE_STATE;
   function NEXT(Q: in STATE_TYPE;
                 I: in INPUT_TYPE)
                return STATE_TYPE;
   function OUTPUT(Q: in STATE_TYPE;
                   I: in INPUT_TYPE)
                return OUTPUT_TYPE;
begin
   P1: Q <= NEXT(Q, I);    -- sensitive to Q, I
   P1: O <= OUTPUT(Q, I);  -- sensitive to Q, I
end;
```

Figure 5. A $\overline{M}$ program that ought to be *M*

which a cycle-level abstraction can exist in an arbitrary VHDL program. Full abstraction was shown to be the condition when the cycle-level model $\mathscr{M}$ is a submodel of the non-abstract one $\mathscr{M}_\delta$. This embedding required that the VHDL programs be *RMC*. Yet unrestricted discrete-event semantics is $R\overline{M}\overline{C}$ / $\overline{R}_\delta M_\delta C_\delta$. Thus admissible Synchronous VHDL programs are identified through *modularity-checking* and *causality-checking* problems. Programs describing acyclic logic networks are trivially admissible.

Acknowledgments

This work was sponsored by the U.S. Defense Advanced Research Projects Agency and monitored by the U.S. Department of Justice Federal Bureau of Investigation under contract DABT-63-95-C-0074. We also thank Gérard Berry for the use of his Esterel compilers and his thoughtful conversations with the first author during his visit to École des Mines de Paris in Spring 1994.

References

[1] W.C. Baker and A.R. Newton, "An Application of a Synchronous Reactive Semantics to the VHDL Language," In *Proc. of 6th IWHLS*, D.D. Gajski editor, 1992.

[2] W.C. Baker, *An Application of a Synchronous/Reactive Semantics to the VHDL Language*, M.S. Report, Dept. of EECS, University of California, Berkeley, USA, January 1993, UCB/ERL M93/10.

[3] W.C. Baker, *Interfacing System Description Languages to Formal Verification*, Ph.D. Dissertation, Dept. of EECS, University of California, Berkeley, February 1996.

[4] G. Berry and G. Gonthier, "The Esterel Synchronous Programming Language: Design, Semantics, Implementation," In *Science of Computer Programming*, Vol.19, No.2, 1992, pp 87-152.

[5] J. Bormann, J. Lohse, M. Payer and G. Venzl, "Model Checking in Industrial Hardware Design," In *Proc. of 32nd DAC*, June 1995, pp 298-303.

[6] K. Brace, R. Rudell and R. Bryant, "Efficient Implementation of a BDD Package," In *Proc. of 27th DAC*, June 1990, pp 40-45.

[7] *Esterel V3*, CIS INGENIERIE, Agence Provence Est, Les Cardoulines, B1 06560 Valbonne, France, 1990.

[8] V. Chaiyakul, D.D. Gajski and L. Ramachandran, "High-Level Transformations for Minimizing Syntactic Variances," In *Proc. of 30th DAC*, June 1993, pp 413-418.

[9] M. Chiodo, L. Lavagno, H. Hseih, K. Suzuki, A. Sangiovanni-Vincentelli and E. Sentovich, "Synthesis of Software Programs for Embedded Control Applications," In *Proc of 32nd DAC*, June 1995, pp 587-592.

[10] A. Debreil and P. Odda, "Synchronous Designs in VHDL," In *Proc. of EuroDAC/EuroVHDL*, September 1993, pp 486-491.

[11] D.D. Gajski, F. Vahid, S. Narayan and J. Gong, *Specification and Design of Embedded Systems*, Prentice Hall, 1994

[12] C.A. Gunter and D.S. Scott, "Semantic Domains," In *Handbook of Theoretical Computer Science*, Vol.B, The MIT Press, 1990, pp 635-675.

[13] N. Halbwachs, *Synchronous Programming of Reactive Systems*, Kluwer Academic Publishers, 1993.

[14] C. Huizing and R. Gerth, "Semantics of Reactive Systems in Abstract Time," In *Real-Time: Theory in Practice*, J.W. de Bakker, C. Huizing, W.P. de Roever, and G. Rozenberg, editors, Proc. of REX Workshop, June 1991, pp 291-314.

[15] *IEEE Standard VHDL Language Reference Manual*, The IEEE, 1987, 345 East 47th Street, New York NY 10017, USA, Std 1076-1987.

[16] *IEEE Standard VHDL Language Reference Manual*, The IEEE, 1993, Std 1076-1993.

[17] T. Kam and P.A. Subrahmanyam, "Comparing Layouts with HDL Models: A Formal Verification Technique," In *Proc. of ICCD*, October 1992, pp 558-591.

[18] Z. Kohavi, *Switching and Finite Automata Theory*, McGraw-Hill, 1978.

[19] J. Lohse, J. Bormann, M. Payer and G. Venzl, *VHDL-Translation for BDD-Based Formal Verification*, Technical Report, Siemens AG, 1994.

[20] D. Knapp, T. Ly, D. MacMillen and R. Miller, "Behavior Synthesis Methodology for HDL-Based Specification and Validation," In *Proc. of 32nd DAC*, June 1995, pp 280-291.

[21] S. Malik, "Analysis of Cyclic Combinational Circuits," In *Proc. of ICCAD*, November 1993, pp 618-625.

[22] P.C. McGeer, K.L. McMillan, A. Saldanha, A.L. Sangiovanni-Vincentelli and P. Scaglia, "Fast Discrete Function Evaluation using Decision Diagrams," In *Proc. of 32nd DAC*, June 1995, pp 402-407.

[23] K. Mulmuley, *Full Abstraction and Semantic Equivalence*, Ph.D. Thesis, Carnegie Mellon University, 1986; Also available from The MIT Press, 1986.

[24] T.R. Shiple, G. Berry and H. Touati, "Constructive Analysis of Cyclic Circuits," In *Proc. of ED&TC*, March 1996.

[25] J.E. Stoy, *The Scott-Strachey Approach to Programming Language Theory*, The MIT Press, 1977.

[26] F. Vahid and D.D. Gajski, "Obtaining Functionally Equivalent Simulations Using VHDL and A Time-Shift Transform," In *Proc. of ICCAD*, November 1991, pp 362-365.

[27] *Verilog Hardware Description Language Reference Manual (LRM)*, November 1991, Open Verilog International, Suite 408, 1016 East El Camino Real, Sunnyvale CA 94087, USA.

[28] R.E. Ziemer, W.H. Tranger and D.R. Fanin, *Signals and Systems: Continuous and Discrete*, Macmillan Publishing, 1989.