BDD-Based Testability Estimation of VHDL Designs*

Fabrizio Ferrandi # Franco Fummi # Enrico Macii ‡ Massimo Poncino ‡ Donatella Sciuto #

Politecnico di Milano
Dip. di Elettronica e Informazione
Milano, ITALY 20133

‡ Politecnico di Torino
Dip. di Automatica e Informatica
Torino, ITALY 10129

Abstract

In this paper we present a method, based on symbolic ATPG techniques, that allows the designer to predict the testability of of a control-oriented complex design specified as a set of interacting VHDL modules. Conversely from existing approaches, our method is purely functional, that is, it does not subsume the knowledge of a gate-level implementation of the system being analyzed. Therefore, it allows us to compute testability estimates with a high degree of accuracy for examples on which existing tools fail due to the enormous amount of information they have to handle when considering the structural implementation of the circuit under investigation. Preliminary experimental results demonstrate the effectiveness of the proposed technique.

1 Introduction

The ability of accurately estimating the testability of a digital system at the early stages of the development process, that is, before the behavioral specification is synthesized into a gate-level implementation, is very important in modern VLSI circuit design. In fact, it may help reducing the total system development time, as well as increasing the quality of the final realization of the chip with respect to some cost functions, such as area, speed, power consumption, and testability.

In industrial environments, the synthesis of highly complex devices is based on the specification of networks of interconnected finite state machines controlling data paths (IFSMD [1]) using hardware description languages such as VHDL.

In this paper, we propose a technique to estimate the fault coverage of a design before the logic synthesis phase, thus allowing an evaluation of the quality of the VHDL code in terms of testability. This information enables the designer to decide whether or not the application of Dft to the VHDL source code is required to eventually achieve testability improvements in the final circuit realization. The fundamental observation here is that an early modification of the design is always beneficial from several points of view, such as area, performance and power consumption of the gate-level circuit, as well as time savings in completing the system development.

*This work was supported, in part, by OMI/REQUEST n. 20616 – Reuse and Quality Estimation.

Our contribution is the first outcome of the testability-related portion of an on-going research project* whose ultimate objective is the definition and implementation of a VHDL testability quality checker. This in the more general context of the realization of a flexible development framework that allows a fast and right-the-first-time design of easily testable systems, along the realization of the associated testing procedures.

The main problem arising when the goal is evaluating the testability of a VHDL description is the reliability and the accuracy of the predicted test costs that the estimator is able to guarantee. This is due to the fact that it is mandatory to define a fault model on the functional description that must be related to the gate-level one. The solution we consider is *test generation-based*, that is, it relies on the use of symbolic test pattern generation to identify faults in the VHDL code. This is done by way of a comparison between the correct and the faulty systems, being the latter derived from the former through insertion of a fault. Clearly, the inserted fault is specified at the functional-level, but it is strictly related to the corresponding gate-level fault, thanks to the knowledge of the algorithm used to synthesize the VHDL specification into a netlist. This method, differently from others in the literature [3], implicitly evaluates the circuit's controllability and observability without defining quantitative measures; conversely, it establishes the chance of successfully identifying a test pattern for each of the considered functional faults.

Approaches in the literature considering the problem of test generation and testability analysis for VHDL are of two types: Coverage-oriented, whose goal is to guarantee the exercise of all functionalities [4], and fault-oriented, whose objective is to obtain fault coverage information from the behavioral model [5, 6]. Fault-oriented test generation relies on the association of failure modes to the hardware description language constructs [7]. However, the validation of such fault model with respect to the stuck-at fault model depends on the logic synthesis algorithm used, and it cannot be assumed in general. The correlation identified by Ghosh is approximately 80%. Following this modeling strategy, different methods have been proposed [6, 8]. All techniques use a graph-based internal representation of the VHDL description for explicit path sensitization and mainly consider data-path based architectures [9, 10].

EURO-DAC '96 with EURO-VHDL '96
0-89791-848-7/96 $4.00

The method we present is based on the assumption that the system under consideration is described in VHDL as a network of interacting FSMDs. In addition, it is assumed that the testability for the data portion of the system is obtained through scan registers and appropriate driving circuitry [2]; in fact, this is what is commonly done in the development of practical designs. Therefore, the focus is on the control part of the system, whose behavior is usually specified through a network of interacting FSMs (IFSM). In this sense, we can claim that the approach to behavioral VHDL testability estimation we present in this paper is innovative; in fact, solutions to the problem of testing control-oriented architectures proposed in the past [11, 12, 13] always resorted to the use of information coming from the structural implementation of the circuit. On the other hand, the analysis we carry out is purely functional.

As we just said, we are interested in estimating the testability of the control logic by operating directly at the specification level, that is, without resorting to the information that becomes available after logic synthesis. To do this, we extract from the VHDL description the control part constituted by interacting FSMs, and we translate their VHDL descriptions into an internal implicit representation based on BDDs [14]. Then, we assume the control logic to be implemented initially by a two-level network; this allows us to identify from the BDD representation of the VHDL specification the essential prime implicants that will surely be included in any gate-level realization of the design. Stuck-at faults on such implicants represent our functional fault model. The multi-level implementation can then be obtained from the two-level one by applying some testability preserving transformations [15]. Test generation is finally performed for the BDD representations using a tool [16] we have developed in the context of gate-level optimization of interacting FSMs [17]. The testability of each module (FSM) can thus be estimated by using the fault coverage achieved for it.

The remainder of this manuscript is organized as follows. Section 2 introduces the adopted VHDL-level fault model, the procedures to extract from the VHDL source code of a controller the corresponding BDD representation, a set of faults, and the BDD representation of the controller containing one of such faults. Section 3 summarizes the BDD-based test pattern generation algorithm of [16]. Section 4 reports some preliminary experimental results demonstrating the viability and the effectiveness of the proposed approach. Finally, Section 5 is devoted to concluding remarks and directions for future research.

2 Fault Modeling at the RT Level

The adopted fault model is based on a BDD representation of the behavior of each FSM composing the VHDL description of a device. Thus, we first introduce a technique to extract such BDD representation from a VHDL description at the register-transfer (RT) level.

2.1 Extraction of the Global Relation of a FSM from VHDL

The test generation algorithm of [16] builds a global relation, represented as a BDD, for each component of an IFSM starting from its gate-level implementation. Here, we propose to build the same relation, still represented as a BDD, through direct translation of the VHDL code.

Three main reasons advise to follow this strategy. First, testability information becomes available before RT-level synthesis. The designer can evaluate the testability problems, concerning the concatenation of VHDL entities, before their RT-level synthesis. It is fundamental to modify for testability a design from the early stages of its specification. Second, VHDL synthesis tools infer some memory elements during RT-level synthesis that do not concern with the actual behavior. Such memory elements increase the number of states of the implemented controller and also the complexity of the corresponding BDD representation. For instance, let us consider the simple VHDL code reported in Figure 1, describing the behavior of a synchronous controller when it is in state `p0`.

```
CASE state IS
   WHEN p0 =>
        IF start_m = '1' THEN
            state <= refresh;
            end_m <= '0';
            valid_pol <= '0';
            start_r <='1';
        ELSE
            state <= p0;
            valid_pol <= '1';
            start_r <= '1';
        END IF;
```

Figure 1: VHDL Description of a Part of a Controller.

The output signals `end_m`, `valid_pol` and `start_r` are assigned into a synchronous process, thus they must be connected to flip-flops. Moreover, `end_m` is not assigned in the `ELSE` branch of the condition, thus it requires a *default* value to be synthesized. Consequently, the synthesis tool (e.g., [19, 20]) inserts a *muxed-flip-flop* onto such an output port. It is evident that the implemented FSM has an higher number of states with respect to the specification, and most of them are equivalent. Thus, the global relation extracted from the implementation would have an unnecessarily high number of state variables. Finally, unspecified transitions are not included in the global relation. During the extraction of the global relation of a sequential circuit from its implementation, even transitions outgoing from unspecified states are included. In fact, all combinations of input and present state variables are implicitly taken into account to identify the FSM's transitions. On the contrary, the global relation of a controller includes only specified transitions if it is directly constructed from a VHDL specification. A lower number of transitions usually implies a smaller BDD.

The technique we propose to translate a VHDL description specifying a controller into the corresponding relation represented as a BDD starts by analyzing the VHDL architecture associated with the entity of the FSM. Two types of analysis are performed. First, a state identification is carried out. In fact, considering the execution flow of the sequential process, a signal or variable belongs to the state of the FSM when it is read before any write or initialization instruction. After state identification, the global relation is computed. The style of the finite state machine considered presents a process with a conditional structure. Figure 2 reports a simplified description of a FSM.

```
statement_list =
   branch | assignment statement_list
branch =
   <IF> condition <THEN>
       statement_list
   <END IF> |
   <IF> condition <THEN>
       statement_list
   <ELSE>
       statement_list
   <END IF>
```

Figure 2: General VHDL Structure of a FSM.

The VHDL assignment statement is translated into the relation $Assign(x, s, z, t)$ which depends on the primary inputs x, the current state s, the primary outputs z and the next state t. Each signal/variable is coded into a BDD variable, and the high-level operator is converted into an array of Boolean functions represented by BDDs. So, each expression included in an assignment statement is computed by composing its high level operators. Then *Assign* relation is computed by relating the output or the next state variable with the Boolean functions associated with the considered expression. Similarly, the VHDL condition statement is translated into the *Cond(x,s)* relation.
The VHDL branch statement is translated into relation $Branch(x, s, z, t)$ by using the following formula:

$$Branch(x,s,z,t) = \frac{Cond(x,s)}{Cond(x,s)} \cdot Stat\_then(x,s,z,t) + \overline{Cond(x,s)} \cdot Stat\_else(x,s,z,t),$$

where $Stat\_then(x, s, z, t)$ and $Stat\_else(x, s, z, t)$ identify the two relations associated with the `statement_list`s of the `THEN` and the `ELSE` clauses. $Stat\_else(x, s, z, t)$ is equal to 0 when the branch statement does not present the `ELSE` branch.
Finally, the VHDL `statement_list` is translated into the $Stat(x, s, z, t)$ relation by using the following formula:

$$Stat(x,s,z,t) = Branch(x,s,z,t) + Stat\_list(x,s,z,t)$$
or
$$Stat(x,s,z,t) = Assign(x,s,z,t) + Stat\_list(x,s,z,t),$$

where *Stat_list* is different from 0 when `statement_list` presents more than one statement.

The global relation associated with the device is then computed by recursively using the formulas shown above.
As an example, let us consider the description of the part of controller shown in Figure 1. The formula associated with the `IF` statement is:

$$\texttt{start\_m} \cdot (\texttt{refresh} \overline{\oplus} \texttt{state} + \overline{\texttt{end\_m}} + \overline{\texttt{valid\_pol}} + \texttt{start\_r}) + \overline{\texttt{start\_m}} \cdot (\texttt{p0} \overline{\oplus} \texttt{state} + \texttt{valid\_pol} + \texttt{start\_r}).$$

2.2 Fault Model

Let us consider a n-input, single-output Boolean function g. A minimal two-level implementation of g is composed of *essential primes* and *primes* that together cover the on-set of the function with minimum cost. Let p_e be the set of essential primes, $|p_e|$ its cardinality, and k_e the total number of its literals. Similarly, let p be one set of primes required to complete the minimum cost cover, $|p|$ its cardinality, and k the total number of its literals.
All the stuck-at faults of the implementation of function g are guaranteed to be equivalent to the faults belonging to the following classes [2]:

- SA-1 faults on all literals of each prime in $p_e \cup p$ (i.e., they are equivalent to SA-1 faults on the *and* and *or* gates).
- SA-0 faults on *one* literal of each prime in $p_e \cup p$ (i.e., they are equivalent to SA-0 faults on the *and* and *or* gates).
- SA-0 and SA-1 faults on all input variables (i.e., they represent stuck-at faults on the *inverter* gates by assuming that all input variables are in complemented form for at least one prime in $p_e \cup p$).

The upper-bound U_g on the total number of faults is:

$$U_g = k_e + k + |p_e| + |p| + 2n.$$

Clearly, faults corresponding to the essential primes are present in every implementation of the Boolean function, independently on the other primes selected to complete the cover (i.e., the primes in p). A necessary condition to test all stuck-at faults of every implementation of g is then the test of faults corresponding to the essential primes. Therefore, the number L_g of stuck-at faults that always need to be considered is:

$$L_g = k_e + p_e + 2n.$$

The considerations above can be extended to the case of a n-input, m-output Boolean function $G = (g_1, \ldots, g_m)$ by letting p_e be the union of the p_e^i sets of essential primes of each single output function g_i composing G. Furthermore, there are multi-level transformations of two-level implementations which preserve testability [15]. Test patterns generated for a two-level implementation can then be extended to a multiple-level one, by applying such transformation techniques.

2.3 Generation of the Faulty Global Relation of a FSM

The test generation approach presented in Section 3 is based on the comparison between the fault-free global relation of a FSM, M, represented as a BDD, and the faulty global relation of the FSM, M_F, still represented as BDD, obtained from M by injection of one fault. Let $R(x,z,s,t)$ be the global relation of M and let $R_F(x,z,s,t)$ be the global relation of M_F.

In this section we describe how relation R_F can be generated starting from R for a given fault f. To do so, we separately consider each Boolean function $R_i(x,s)$, associated with each primary output z_i and state output t_i, so that $R(x,z,s,t) = \prod R_i(x,s) \overline{\oplus} b_i : b_i = z_i | b_i = t_i$.

The essential primes of the Boolean function representing the global relation R are implicitly computed by applying the algorithm in [18]. The faulty global relation, R_F, is then determined based on the set of primes $(p_e \cup p)$ in the two-level cover, and on the previously described three classes of faults. In fact, the following three different strategies for fault insertion are applied in relation to the type of fault.

- SA-1 faults on all literals of each prime in $p_e \cup p$.
  Let $ip(x_1, \cdots, x_l, s_1, \cdots, s_m) \in p_e$ be an essential prime, and $b_i : b_i = x_i | b_i = s_i$.
  $\forall b_i : ip(x,s)_{b_i} \neq ip(x,s)_{\overline{b_i}}$ (i.e., $ip(x,s)$ depends on b_i) then
  the faulty prime $ip_F(x,s) = ip(x,s)_{b_i}$ and
  the faulty relation $R_{i_F}(x,s) = R_i(x,s) \cdot \overline{ip(x,s)} + ip_F(x,s)$
  (i.e., the fault-free essential prime is replaced by the corresponding faulty prime).
- SA-0 faults on *one* literal of each prime in $p_e \cup p$.
  Let $b_i : b_i = x_i | b_i = s_i$.
  $\exists b_i : ip(x,s)_{b_i} \neq ip(x,s)_{\overline{b_i}}$ then
  the faulty prime $ip_F(x,s) = ip(x,s)_{\overline{b_i}}$ and
  the faulty relation $R_{i_F}(x,s) = R_i(x,s) \cdot \overline{ip(x,s)} + ip_F(x,s)$.
- SA-0 and SA-1 faults on all input variables.
  Let $b_i : b_i = x_i | b_i = s_i$.
  $\forall b_i : R_i(x,s)_{b_i} \neq R_i(x,s)_{\overline{b_i}}$ then the faulty relation
  $R_{i_F}(x,s) = R_i(x,s)_{\overline{b_i}}$ for SA-0 fault and
  $R_{i_F}(x,s) = R_i(x,s)_{b_i}$ for SA-1 fault.

At the end, the global faulty relation, $R_F(x,s)$, is reconstructed starting from each computed faulty relation $R_{i_F}(x,s)$.

3 Testing Strategy

In [16], we presented a testing strategy for complex systems structured as a network of FSMs. The method considers FSM networks whose connection graphs are DAGs, since it can be shown that, once a specific FSM, M, has been selected, it is always possible to find a serial decomposition, i.e., a *topological sort*, of such network with respect to M, even in presence of reconvergent fanout connections.

The set of FSMs driving M originates the *controlling network* (C), while the set of FSMs driven by M originates the *observing network* (O). As an example, Figure 3 shows the controlling and the observing networks for FSM_3.

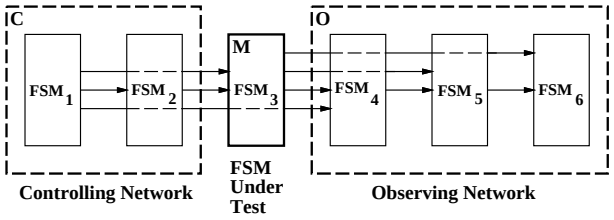

Figure 3: Controlling and Observing Networks for FSM_3.

The testing procedure works in three steps. First, after injecting fault f into M, the set TS_M of all minimum length test sequences for fault f are computed. If no sequence exists, the fault is untestable. In the second phase, it is verified whether the controlling FSM, C, is able to generate some of the sequences in TS_M. If this is the case, a sequence JS_C, representing the justification sequence of TS_M is generated; otherwise, the fault f is marked as non-excitable. In the third phase, it is verified if, under application of the test sequence $(JS_C + TS_C)$, the fault can be observed on the primary output lines of machine O. To do this, we first fault simulate the test sequence $(JS_C + TS_C)$ to obtain the corresponding sequences JS_O and TS_O, in terms of the input variables of O. Then, we determine the set of observing states of O for which the outputs of O differ, under the application of the distinguishing vectors of TS_O. If no observing state exists or is reachable, then fault f is non-observable with respect to set of sequences that have been considered. Otherwise, the set of state of O that originate sequences $JS_O + TS_O$ is determined; if such a set contains the reset state of machine O, then fault f is observable.

For further details about specific steps of the testing algorithm, the reader is referred to [16].

4 Experimental Results

In this section we present preliminary results we have obtained by applying the testability estimation techniques proposed in this paper to some benchmark designs.

The first set of data, shown in Table 1, aims at demonstrating the advantages that one may get in extracting the BDD-based description of a FSM directly from the VHDL code instead of from a gate-level implementation. The considered circuits come from both industrial and academic environments. Those with the `_ctrl` suffix are parts of the *vending machine* reported in the VHDL book of Perry [21] and the others are telecom controllers. For each circuit we present the total number of primary inputs and primary outputs; then, the number of states and the number of BDD nodes of the global relation of the FSM associated to the circuit in the case such relation is extracted from the VHDL source code; on the other hand, when the gate-level netlist is used, we report the number memory elements and, again, the number of BDD nodes of the global relation of the FSM associated to the circuit. This comparison clearly highlights what is the advantage one can get by using the direct extraction technique of this paper.

Example	*Inputs*	*Outputs*	*VHDL*		*Netlist*	
			States	*BDD Nodes*	*FF*	*BDD Nodes*
`change_tim_ctrl`	5	7	32	131	10	4999
`coin_tim_ctrl`	5	3	16	61	5	234
`exac_change_ctrl`	10	2	65536	431	17	2079444
`item_tim_ctrl`	10	4	2	43	5	531
`write`	10	8	7	178	11	12610
`tx`	10	10	44	380	16	264327
`itg`	7	5	8	140	3	124
`alfa`	7	6	14	237	4	193

Table 1: Results: Extraction of the Global Relation From VHDL and Netlist.

In the case of benchmarks named `itg` and `alfa`, the BDD representations of the global relations extracted from the netlists are a slightly smaller than the BDD representations extracted from the VHDL source code. This depends on the different degrees of specification of the implemented circuits and the corresponding VHDL descriptions, that sensibly affect such small examples. On the contrary, the other global relations extracted directly from VHDL are significantly smaller than the corresponding BDD representations extracted from the netlists. Consider, for instance, circuit `exac_change_ctrl`. For this example, only 431 BDD nodes are required to represent its global relation when the starting point is the VHDL description. Such number becomes almost unmanageable (i.e., memory occupation gets very close to the machine limit) when the extraction is performed starting from the netlist.

The second set of data that we present concerns the analysis of the accuracy that the proposed testability estimator is able to guarantee.

To carry out the experiments we have chosen an industrial chip realizing the functionalities of a telecommunications controller. The design is composed of several interconnected FSMDs. All data-path sections have been omitted, since we assume that they are testable (possibly by applying a scan-path technique). The control logic, on the other hand, originates the IFSM depicted in Figure 4 which contains five components. A possible implementation of this circuit contains a total of 37 synchronous memory elements (i.e., flip-flops) and approximately 600 combinational logic gates.

In Table 2 we compare, for each component of the IFSM, the fault coverage obtained by applying the test generation procedure of [16] to the BDD representations of the FSMs global relations extracted from the VHDL descriptions and the gate-level netlists.

Comp	*VHDL*			*Netlist*	
	Faults	*FC (%)*	*CPU (s)*	*Faults*	*FC (%)*
FSM1	74	85.1	474.6	53	83.0
FSM2	151	54.3	414.9	137	57.6
FSM3	186	7.9	88.3	198	8.7
FSM4	82	97.6	850.7	72	94.5
FSM5	248	95.0	971.0	226	98.0

Table 2: Results: Fault Coverage Comparison.

When the VHDL source code is used, only the faults corresponding to the essential primes of the two-level cover have been taken into account, and they have been modeled without resorting to the structural circuit description. On the other hand, when the gate-level implementation is used, all the faults present in the FSM implementation have been considered. The two coverage are very similar, thus underlining the effectiveness of the proposed approach.

It is interesting to notice that gate-level test generation can not be performed on the design implementation using available structural (e.g., HITEC [22]) or traversal-based (e.g., Veritas [23]) test generators, due to the size of the circuit to be handled. On the contrary, the test generator presented in [16] identifies the nature (i.e., testable or redundant) of each stuck-at fault in the circuit implementation. Moreover, information regarding redundant faults identified at the VHDL level allows a designer to modify the device specification before its actual synthesis. In this example, it is clear that `FSM3` is difficult to be tested due to its interconnections.

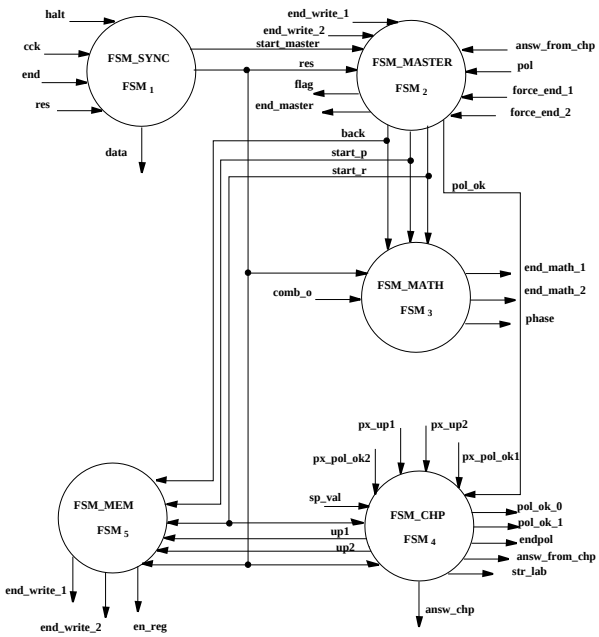

Figure 4: IFSM Description of a Telecom Device.

5 Concluding Remarks

The complexity of modern VLSI systems no longer allows the development of new designs without the help of automatic synthesis tools. Once the behavioral specification of the system is available, human intervention is required to guide the synthesis process by means of precise architectural and technological choices. Among the quality criteria to be considered while designing a new chip, testability plays a central role; this is why sophisticated design for testability techniques have been proposed in the past. Clearly, achieving high testability may be expensive in terms of chip area, speed, and power consumption; therefore, the use of DfT techniques should be limited to the cases where no alternatives exist.

In order to reach this goal is thus essential to be able to estimate the testability of the system under development as early as possible in the design process. This is because, at the high level of abstraction, the complexity of the description is still manageable, since several implementation details are not relevant and, therefore, can be kept hidden. In addition, making the right architectural choices may result in large advantages from the view-point of the quality of the circuit produced by the logic synthesis tool.

The main contribution of this paper is a technique to predict the testability of a control-oriented digital design described in VHDL at the behavioral level. In particular, the system is assumed to be composed of several interacting modules, being this the way industrial circuits are designed. A BDD-based representation of each module is then extracted from the corresponding VHDL source code. Faults, modeled at the functional level, are then injected, one at a time, to create a faulty instance of each module, still represented as a BDD. Test generation on the resulting network of interacting modules is performed and some global testability information retrieved.

All the proposed algorithms are fully symbolic, that is, they completely rely on BDD-based, implicit enumeration techniques to handle a large amount of information; as a consequence, reasonably large examples can be processed by our testability estimation tool in very short times. Experimental results, though preliminary, are very promising, and prove the viability of the approach presented here.

As future work, we are currently looking into the problem of extending the proposed technique to the case of designs whose behaviors can be modeled by generic FSMDs. This in the more general context of building a quality checker and cost predictor for complex digital systems described in VHDL at the RT level.

References

[1] D. Gajski, N. Dutt, A. Wu, S. Lin, High-Level Synthesis: Introduction to Chip and System Design, Kluwer Academic Publishers, 1992.

[2] M. Abramovici, M. A. Breuer, A. D. Friedman, Digital Systems Testing and Testable Design, Computer Science Press, 1995.

[3] X. Gu, K. Kuchcinski, Z. Peng, "Testability Analysis and Improvement from VHDL Behavioral Specifications," EuroVHDL'94: IEEE European VHDL Conference, pp. 644-649, Grenoble, France, September 1994.

[4] S. R. Rao, B. Y. Pan, J. R. Armstrong, "Hierarchical Test Generation for VHDL Behavioral Models," EuroVHDL'94: IEEE European VHDL Conference, pp. 175-182, Hamburg, Germany, September 1993.

[5] U.H. Levendel and P.R. Menon. "Test Generation Algorithms for Computer Hardware Description Languages," IEEE Transactions on Computers, Vol. C-31, No. 7, pp. 557-588, July 1982.

[6] N. Giambiasi, J. F. Santucci, A. L. Courbis, V. Pla, "Test Pattern Generation for Behavioral Descriptions in VHDL," EuroVHDL'94: IEEE European VHDL Conference, pp. 228-235, Hamburg, Germany, September 1991.

[7] S. Ghosh, T. J. Chakraborty, "On Behavior Fault Modeling for Digital Designs," Journal of Electronic Testing: Theory and Applications, Vol. 2, pp. 135-151, 1991.

[8] V. Pla, J. F. Santucci, N. Giambiasi, "On the Modeling and Testing of VHDL Behavioral Descriptions of Sequential Circuits," EuroVHDL'94: IEEE European VHDL Conference, pp. 440-445, Hamburg, Germany, September 1993.

[9] P. Vishakantaiah, J. Abraham, M. Abadir, "Automatic Test Knowledge Extraction from VHDL ATKET," DAC-29: ACM/IEEE Design Automation Conference, pp. 273-278, Anaheim, CA, June 1992.

[10] P. Vishakantaiah, J. A. Abraham, D. G. Saab, "CHEETA: Composition of Hierarchical Sequential Tests Using ATKET," ITC'93: IEEE International Test Conference, pp. 606-615, Baltimore, MD, October 1993.

[11] A. Ghosh, S. Devadas, A. R. Newton, "Sequential Test Generation and Synthesis for Testability at the Register-Transfer and Logic Levels," IEEE Transactions on CAD/ICAS, Vol. CAD-12, No. 5, pp. 579-598, May 1993.

[12] K. T. Cheng, "An ATPG-Based Approach to Sequential Logic Optimization," ITC'91: IEEE International Test Conference, pp. 372-375, Nashville, TN, October 1991.

[13] U. Glaser, K. T. Cheng, "Logic Optimization by an Improved Sequential Redundancy Addition and Removal Technique," IEEE Asia South-Pacific Design Automation Conference, pp. 235-240, Chiba, Japan, August 1995.

[14] R. Bryant, "Graph-Based Algorithms for Boolean Function Manipulation," IEEE Transactions on Computers, Vol. C-35, No. 8, pp. 79-85, August 1986.

[15] S. Devadas, A. Ghosh, K. Keutzer, Logic Synthesis, McGraw-Hill, 1994.

[16] F. Ferrandi, F. Fummi, E. Macii, M. Poncino, D. Sciuto, "Test Generation for Networks of Interacting FSMs Using Symbolic Techniques," GLS-VLSI'96: IEEE Great Lakes Symposium on VLSI, pp. 208-213, Ames, IA, March 1996.

[17] F. Ferrandi, F. Fummi, E. Macii, M. Poncino, D. Sciuto, "Symbolic Optimization of FSM Networks Based on Sequential ATPG Techniques", DAC-33: ACM/IEEE Design Automation Converence, pp. 467-470, Las Vegas, NV, June 1996.

[18] O. Coudert, J. C. Madre, "Implicit and Incremental Computation of Primes and Essential Primes of Boolean Functions," DAC-29: ACM/IEEE Design Automation Converence, pp. 36-39, Anaheim, CA, June 1992.

[19] Synopsys User's Manual, Synopsys Inc., 1994.

[20] Autologic VHDL Reference Manual, Mentor Graphics, 1993.

[21] D. L. Perry, VHDL, McGraw-Hill, 1993.

[22] T. Niermann, J.* Patel, "HITEC: A Test Generation Package for Sequential Circuits," EDAC'91: IEEE European Conference on Design Automation, pp. 214-218, Amsterdam, The Netherlands, February 1991.

[23] H. Cho, G. D. Hachtel, F. Somenzi, "Redundancy Identification/Removal and Test Generation for Sequential Circuits Using Implicit State Enumeration," IEEE Transactions on CAD/ICAS, Vol. CAD-12, No. 7, pp. 935-945, July 1993.