Structural BIST Insertion Using Behavioral Test Analysis[†]

M. Nourani[‡] C. Papachristou[§]

Abstract

The purpose of this work is to develop a test synthesis technique based on BIST methodology which uses the test metrics (i.e. controllability and observability) obtained by test analysis of the behavior to enhance the testability quality (fault coverage) of the corresponding structure and obtain the scheduled test behavior accordingly. The key feature of this work is in using the Structured Data Flow Graph *(SDG) which annotates the behavioral information (e.g. data dependency) and structural information (e.g. binding, connectivity). To enhance testability of the structure, the SDG will be modified using transformation technique to improve the fault coverage and shorten the test schedule.*

1. Introduction

The design complexity of microelectronic chips continues to increase. Testing of these complex chips is becoming prohibitively expensive in regards to equipment and time. There are a great number of alternative test methodologies such as built-in self testing (BIST), which can be performed at circuit speed.

Although there are many advanced design synthesis tools that provide efficient CAD environments at several levels of the design hierarchy, few support Design for Test (DFT) even as a post-synthesis process. The few tools providing DFT automation operate at the logic level of the hierarchy. Our work is motivated by two main issues: a) the need to expand the test hierarchy to higher levels, particularly the behavioral level, in coordination with the structural design process; b) the benefits of Built-In Self Test (BIST) as a DFT methodology for insertion in large, high-speed designs.

Several testability metrics at the behavioral level based on the notion of entropy have been introduced by researchers [ThAb89]. Entropy of a binary signal X is defined as: $I_X = \sum_{i=0}^{2^{|X|}-1} p_{X,i} \log_2 \frac{1}{p_{X,i}}$, where $|X|$ and $p_{X,i}$ denote the bit width and the probability that X is in state i ($0 \leq i \leq 2^{|X|} - 1$), respectively. These metrics are used to evaluate testability at an early design stage [ChLP92] [ChKS94] [HaOr94]. The metrics quantify how the quality of pseudo-random values deteriorates as the values propagate through intermediate variables, and the sensitivity of the behavior to erroneous values, i.e., how easily an erroneous value is propagated to an observable point.

The *controllability* of a point directly relates to the randomness quality of the test patterns generated at that point. The *randomness* metric [ThAb89] [ChPa91] is the ratio between the entropy of a signal and the maximum entropy coming from an ideal random pattern generator. Based on this definition, the randomness ranges from 0 for a signal taking a single value to 1 for a perfect random signal. The *observability* of a signal is the accuracy by which we can tell what is happening to that signal in terms of faulty value by checking the primary outputs of the behavior and without looking at the signal itself. The *transparency* is the the probability that a fault effect (an arbitrary change in the signal value) can be observable at least in one of the primary outputs [ChPa91]. Transparency ranges from 0 for signals that can not be observed at all, to 1 for signals that can be directly observed.

There are at least three techniques for analyzing testability of entire behaviors and structures consisting of embedded modules: (i) Transformation-based analysis [DePo94] [GCRD94] [Peng94] [PeKu94]; (ii) Markov model [CaPa95] [ChGu89]; (iii) Monte Carlo simulation [CaPa95]. Usually, these test metrics are derived by exact analysis and/or simulation and a comprehensive library of non-embedded modules is constructed. This library database will be used as the basis of a heuristic for analyzing the entire behaviors or structures with embedded modules.

After analysis of the entire behavior, appropriate decision can be made to enhance testability (e.g. increasing the fault coverage), e.g. adding a new input and output to play the role of a TPGR and MISR (in the structure to be synthesized), respectively; or to improve the testability of an existing structure [Avra91][LeJW93]. This process is called BIST insertion. The importance of test time [AbBF90] as a component of chip cost has been recently considered as a design attribute [HaOr94]. Careful BIST insertion can also optimize the test time without sacrificing the test quality, i.e. fault coverage.

The main contribution of our work is in using behavioral test analysis to do structural BIST insertion. By using a structured dataflow graph (SDG) model, we present a novel and unified view of both behavior (i.e. DFG) and structure (i.e. datapath) to develop a methodology of enhancing the test quality of an existing circuit produced by a high level synthesizer, automatic or manual. Note that a similar concept, but in completely different context, was introduced in [LePa92] and [ChLP92] for microprocessor testing and testability extraction, respectively. There are two additional advantages of our method: 1) our test scheme ensures that the *entire* circuit structure will be tested by patterns following paths dictated by the test behavior; and 2) we avoid re-exercising the self-loop paths in the structure and thus implicitly break such loops in the test mode without additional overhead.

This paper is organized as follows: Section 2 presents the test-behavior concept. The SDG model and our test methodology are discussed in Section 3. Section 4 describes the transformations and the process of modifying SDG. Experimental results are shown in section 5. Finally, concluding remarks are in section 6.

2. Behavioral BIST Model

The benefits of doing test insertion at the behavioral level are quite similar to those of doing synthesis at the behavioral level. Specifically, at the behavioral level, the test complexity of large designs becomes more manageable, the overall

[†]This work was partially supported by the Semiconductor Research Corporation (SRC) under Contract No. DJ-527.

[‡]Department of Electrical and Computer Engineering, University of Tehran, Tehran 14395, Iran.

[§]Department of Computer Engineering, Case Western Reserve Univ., Cleveland 44106, Ohio.

ED&TC '97 on CD-ROM

design time is reduced, design and test tradeoffs can be performed effectively and it can perform *at-speed* testing. More importantly, a combined design and test behavior can be synthesized by any synthesizer and the result will be a testable design no matter of the specification of the synthesizer. Unfortunately, there is a drawback in using such a universal test-synthesis methodology that is excessive hardware. In other words, all aspects and characteristics of the synthesizer that will be used to generate the final testable architecture is ignored. This may impose excessive hardware (e.g. extra TPGR) to the structure.

Consider a behavior **B** expressed in behavioral VHDL and/or by a data flow graph (DFG) description. For a *minimal behavioral BIST scheme*, we assume that the input and output variables of **B** will be mapped into BIST registers (TPGR/LFSRs* and MISRs†, respectively) at the structural level, using a synthesis tool. **B** has two modes of operation, a normal mode and a test mode. The scheme is *minimal* in the sense that no part of the behavior other than the I/O variables is affected in test mode; there is no BIST insertion within **B** itself. The behavioral test mode is "mapped" into a structural test mode to test the synthesized RTL structure from the behavior.

In this model, we measure the testability of variables and signals based on the controllability and observability metrics discussed earlier. We define a behavior **B** to be *testable* if all its internal variables and signals have randomness values above the threshold *R_tsh* and transparency values above the threshold *T_tsh*. Low randomness or transparency indicates potential controllability or observability problems that may negatively impact the testability of the RTL structure synthesized from the behavior. If **B** is not testable, we augment its testability by using an insertion technique at the behavioral or structural levels. This insertion is equivalent to introducing new primary inputs/outputs (new input/output variables and edges in the DFG and new input/output register, e.g. TPGRs and MISRs, in the structure).

The motivation for defining the notion of test behavior stems from the elevation of the test mode of a circuit to the behavioral level. The test behavior expresses the behavior of a circuit in test mode; this behavior is generally different from the normal mode behavior. We make a distinction between the design behavior **D_B**, i.e., the normal-mode behavior of the circuit, and the test behavior **T_B**. The test behavior **T_B** and design behavior **D_B** are combined into a *design-and-test behavior* **D&T_B**. This combined behavior is "switchable" by nature; it locally switches between the design behavior **D_B** in normal mode and the test behavior **T_B** in test mode. When the design-and-test behavior **D&T_B** is synthesized, the result is a testable circuit, including a test application scheme (test control). This concept is pictured in Fig. 1.

3. Structured Data Flow Graph

The basis of our work is a new special graph, called *Structured Data Flow Graph* (SDG), that annotates the information of the CDFG (e.g. schedule) and the structure (e.g. datapath) simultaneously and can be automatically derived from the CDFG and its corresponding synthesized structure. The importance of SDG lies in its capability to fully characterize both system architecture and behavioral information. SDG contains the following elements:

1. *ALU nodes* corresponding to the operations in DFG and their operators in the structure.
2. *Storage nodes* corresponding to the variables in DFG and their storage elements (e.g. registers).
3. *Data flow edges* corresponding to the data flow in DFG and their interconnect net list.
4. *Attributes* of nodes and edges. Attributes of an ALU node and the storage node are the operation assigned to that ALU and the variable stored in the storage element, respectively. The attribute of an edge is a set of all hardware components in the structure (e.g. MUXes, BUSes, wires) that contribute in transferring data from one point (i.e. ALU or storage node) to another.

Fig. 2 shows the SDG for a small example whose DFG and datapath are available. Note that for simplicity we have shown only the attribute of edges which play a major role in SDG modification process. SDG is topologically equivalent to a DFG whose edges are augmented (by registers holding data) and structural components that contribute in transferring data as their attributes.

The SDG has important features from the testability point of view including: 1) The structure-SDG relation is one-to-many. Given a SDG, only one structure can be constructed from it while given a structure, we can obtain many SDG corresponding to different behaviors. 2) By counting the instances of the datapath components in the SDG, repeated as ALU nodes, storage nodes and elements in the edge attribute sets, we can obtain an indication of activity of the components in the structure. This factor (number of datapath instances in the SDG) will be used to avoid re-exercising the components in the test mode since re-exercising may not only prolongs the test session but also it may deteriorate the quality of the signals (in terms of controllability and observability) propagated in the datapath and consequently may lower the test quality.

3.1 SDG-for-Test Transformations

As discussed earlier, we will modify the SDG based on the test metrics analysis and suggest some local modification in the structure. The SDG modification will be performed based on a transformation technique. Different transformation techniques have been extensively used in the literature [DePo94]. So far, we have identified four transformations to enhance the test quality of the structure shown implicitly in the SDG.

1. *Remove Node (T_{remove_node})*: If there are two or more ALU nodes corresponding to the same ALU with the same attributes for input and output edges (which mean they read the same registers and write into the same register through the same path), keep only one of them and remove the rest. From the testing point of view, only one of them in SDG can exercise all components and connections and the rest are redundant. Re-exercising them does not add any test information while the drawback is the possibility of deteriorating the quality of pseudorandom patterns as they propagate.
2. *Connect Edge ($T_{connect}$)*: Based on the test-metrics analysis, transfer the high randomness signals (above the threshold) to the low randomness point (below the threshold) by establishing a new edge between two points which read the same storage element and cutting the edge which transfers the low randomness.
3. *Remove Empty Step (T_{remove_step})*: After applying transformations T_{remove_node} and $T_{connect}$, the modified SDG usually consists of some empty time steps which prolong the test session without providing any additional information. These empty steps can be removed to shorten the test schedule represented by the SDG.
4. *Move Node (T_{move})*: After applying the above three transformations, further reduction in test schedule time is possible by moving the nodes. If there are such nodes, we will move them to the earlier steps. Some moves require additional registers. So, in performing this transformation a tradeoff between additional registers and

*Test Pattern Generator Register/Linear Feedback Shift Register

†Multiple Input Shift Register

shorter test schedule must be decided. Currently, we apply this transformation only if it does not require any additional register.

4. SDG Modification Using Transformations

In this section, by means of a small example, we show how the transformations are applied to modify a SDG and how the scheduled test behavior is obtained. Fig. 3(a) and (b) show the design behavior (DFG) and the corresponding structure (RTL datapath) to compute $y = ax^4 + b$. The randomness measures (in square brackets) are also given in the figure. Assuming that a randomness threshold of 0.75 and a transparency threshold of 0.8 is acceptable in order to enhance the test quality of the structure, we need to modify SDG and introduce a test behavior.

Fig. 3(c) is the SDG constructed by the process explained in the previous section. Fig. 3(d), (e), (f) and (g) are SDG configurations after applying T_{remove_node}, $T_{connect}$, T_{remove_step} and T_{move}, respectively. The first transformation (see Fig. 3(d)) removes two ALU nodes in time steps 3 and 4 of SDG since keeping them does not add any new information in the test process; they just re-exercise the components and paths. The intent of the second transformation (see Fig. 3(e)) is to enhance the randomness of the adder input. We have two options: enhancing from 0.49 to 0.79 (by connecting the output of the multiplication in the second time step) or 0.93 (by connecting the output of the multiplication in the first time step). The third transformation (see Fig. 3(f)) removes the empty time steps to shorten the test schedule (test time) in the test mode. Finally, the fourth transformation (see Fig. 3(g)) reschedules the addition into time step 2 for further test time reduction. The MSDG and the final scheduled test behavior are shown in Fig. 3(h). Finally, the modified datapath is pictured in Fig. 3(i).

In the example of Fig. 3 there is no need for any datapath modification to enhance testing. In this example, we achieve higher test quality by creating a shorter test behavior while avoiding re-exercising the components.

To show the effect of these modifications, we have implemented these designs (4-bit width) based on the 0.8 micron CMOS library [Vlsi93] within COMPASS [Comp93] on a SPARC-IPC workstation with 32M RAM. The ASIC synthesizer tool in COMPASS takes a datapath circuit description in VHDL and produces a gate-level structure which is later analyzed by AT&T's GENTEST fault simulator [Gent93]. Fault coverage curves corresponding to Fig. 3(c), (f) and (g) reflect the effectiveness of these transformations in enhancing the test quality and are shown in Fig. 4.

5. Experimental Results

The result of testability enhancement of the structure based on the behavioral test metrics for three experiments are tabulated in Table 2. All datapaths for these examples, have been produced by our synthesis tool, *SYNTEST* [Synt92].

The examples are 1) *Polynomial* example used in Sections 2. and 3.; 2) *HAL* example form [PaKn89]; and 3) *Wave filter* [GrTu88] which is the fifth order elliptical wave filter chosen as a benchmark for 1988 High-Level Synthesis Workshop. For comparison, we have implemented these designs (4-bit width) based on the 0.8 micron CMOS library [Vlsi93] within COMPASS [Comp93] on a SPARC-IPC workstation with 32M RAM. The ASIC synthesizer tool in COMPASS takes a datapath circuit description in VHDL and generates a gate level structure and the complete layout, for these benchmarks, in a few minutes. Then, the gate level structure is analyzed by AT&T's GENTEST fault simulator [Gent93].

The layout area in Table 2 shows the datapath area only. The test time reported in the table is the time steps of the test schedule. Before enhancement, the test schedule is assumed to be the same as the design schedule (scheduled DFG) while after the SDG modification, we obtain the scheduled test behavior with the same length (or shorter in all of our experiments).

For the experiments with polynomial and wave filter, reflected in Table 2, there is no hardware overhead. However, generally a tradeoff between test area overhead (due mostly to the extra TPGR's, MISR's and shadow registers) and test time can be made. These two example show that with careful selection of the test schedule by the SDG modification, we can get the same or better fault coverage with no test overhead. The tradeoff between test overhead and test time is reflected in the HAL example in which extra registers are inserted to boost the test quality. The more accurate parameter will be the cumulative time phases (rising and falling edge of the clock signal) for applied random patterns which along with the fault coverage curves are shown in Fig. 4, Fig. 5 and Fig. 6. The horizontal axis shows the number of simulation clocks which is proportional to the number of random patterns in the fault simulation process.

6. Conclusion

A test synthesis technique at the structural level using behavioral information is introduced for BIST methodology. After constructing the SDG graph we use the behavioral test analysis to apply some transformations and modify the SDG to enhance the test quality of the structure while the scheduled test behavior will be generated simultaneously. Our experimental results show that in most cases we can enhance the fault coverage and/or shorten the test time with no test hardware overhead. In other cases, this is achieved by adding extra TPGR, MISR and shadow registers in the structure creating less than 10% overhead in all examples so far attempted.

Our approach based on the test behavior scheme has significant advantages, 1) testing the datapath circuit in its entirety, and 2) avoiding hardware overhead for loop breaking which is normally employed in conventional datapath testing. Another advantage is shortening test time while achieving very high fault coverage.

References

[AbBF90] M. Abramovici, M. Breuer, A. Friedman, *Digital Systems Testing and Testable Designs*, Computer Science Press, 1990.

[Avra91] LaNae Avra, "Allocation and Assignment in High-Level Synthesis for Self-Testable Data Paths," *International Test Conference*, 1991.

[CaPa95] J. Carletta and C. Papachristou, "Testability Analysis and Insertion for RTL Circuits Based on Pseudorandom BIST," *ICCD-95*, pp. 162-167, October 1995.

[ChGu89] C. C. Chuang and A. K. Gupta, "The Analysis of Parallel BIST by the Combined Markov Chain (CMC) Model," *International Test Conference*, pp. 337–343, 1989

[ChKS94] C. Chen, T. Karnic and D. Saab, "Structural and Behavioral Synthesis for Testability Techniques," *IEEE Trans. on CAD*, pp. 777–785, June 1994.

[ChLP92] V. Chickermane, J. Lee and J. Patel, "Design for Testability Using Architectural Descriptions," *International Test Conference*, pp. 752–761, 1992.

[ChPa91] S. Chiu and C. Papachristou, "A Design for Testability Scheme with Applications to Datapath Synthesis," *Design Automation Conf.*, 1991.

[Comp93] Compass Design Automation, "User Manuals for COMPASS VLSI V8R4.4," Compass Design Automation, Inc., 1993.

Table 1: Four SDG-for-test transformations

Name	*Objective*	*Requirement*	*Possible Effects on Structure*
T_{remove_node}	Avoid re-exercising	Attributes equivalency for all inputs/outputs	Same path coverage, Shorter test time
$T_{connect}$	Boost controllability, Boost observability	Attribute equivalency for one input/output	More Path Coverage, Extra Register
T_{remove_step}	Reduce Test Time	Existence of empty steps	Same path coverage, Shorter test time
T_{move}	Reduce Test Time	Creating empty steps	Same path coverage, Shorter test time Extra Register

Table 2: Testability enhancement using test analysis

Design Name	*Before Enhancement*			*After Enhancement*		
	Layout Area $[\mu \times \mu]$	*Test Time [cycle]*	*Fault Coverage*	*Layout Area* $[\mu \times \mu]$	*Test Time [cycle]*	*Fault Coverage*
Polynomial	1218×584	5	88.5%	1218×584	2	100%
HAL	1794×977	7	97.5%	1634×1178	5	98.8%
Wave Filter	2402×1403	21	97.3%	2402×1403	17	98.5%

[DePo94] S. Dey and M. Potkonjak, "Transforming Behavioral Specifications to Facilitate Synthesis of Testable Designs," *International Test Conference*, pp. 184–193, 1994.

[DePo94b] S. Dey and M. Potkonjak, "Non Scan Design-For-Testability of RT-Level Data Paths," *ICCAD-94*, pp. 640-645, Nov. 1994.

[GCRD94] M. Gentil, D. Crestani, A. Rhalibi and C. Durante, "A New High Level Testability Measure: Description and Evaluation," *VLSI Test Symposium*, 1994.

[Gent93] AT&T, "User Manuals for GENTEST_S 2.0," AT&T Bell Laboratories, 1993.

[GrTu88] B. D. Green and L. E. Turner, "New Limit Cycle Bounds for Digital Filters," *IEEE Trans. Circuits and Systems*, April 1988, pp. 365-374.

[HaOr94] I. G. Harris and A. Orailoglu, "Microarchitectural Synthesis of VLSI Designs with High Test Concurrency," *Proc. ACM/IEEE Design Automation Conference*, pp. 206-211, June 1994.

[LeJW93] T. Lee, N. Jha, W. Wolf, "Behavioral Synthesis of Highly Testable Data Paths under the Non-Scan and Partial Scan Environments," *Design Automation Conf.*, 1993.

[LePa92] J. Lee and J. Patel, "An Instruction Sequence Assembling Methodology for Testing Microprocessors," *International Test Conference*, pp. 49–58, 1992.

[PaCa95] C. Papachristou and J. Carletta, "Test Synthesis in the Behavioral Domain," *Internat. Test Conf. (ITC-95)*, October 1995.

[PaKn89] P. Paulin and J. Knight, "Force-Directed Scheduling for the Behavioral Synthesis of ASIC's," *IEEE Trans. on CAD*, June 1989.

[Peng94] Z. Peng, "Testability Driven High Level Synthesis," *Proc. Int. Conf. on ASIC*, 1994.

[PeKu94] Z. Peng and K. Kuchcinski, "Automated Transformation of Algorithms into RTL Implementations," *IEEE Trans. CAD*, pp. 150-165, Feb. 1994.

[Synt92] H. Harmanani, C. Papachristou, S. Chiu and M. Nourani, "SYNTEST: An Environment for System-Level Design for Test," in *Proc. EURO-DAC 92*, Sept. 1992.

[ThAb89] K. Thearling and J. Abraham, "An Easily Computed Functional Level Testability Measure," *International Test Conference*, pages 381–390, 1989.

[Vlsi93] VLSI Technology, "0.8-Micron CMOS VSC450 Portable Library," VLSI Technology, Inc., 1993.

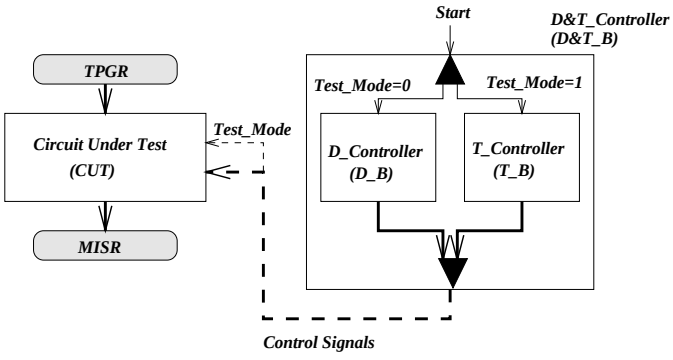

Figure 1: Switchability of design-and-test behavior

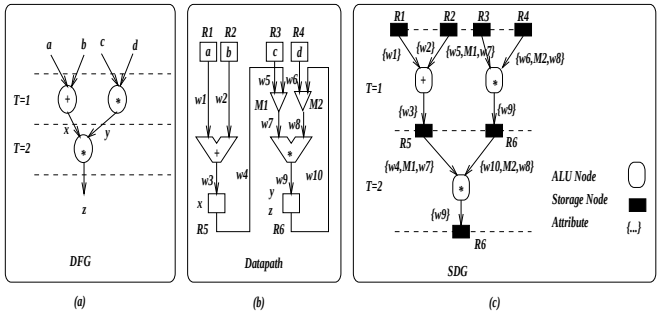

Figure 2: Constructing the SDG from DFG and datapath

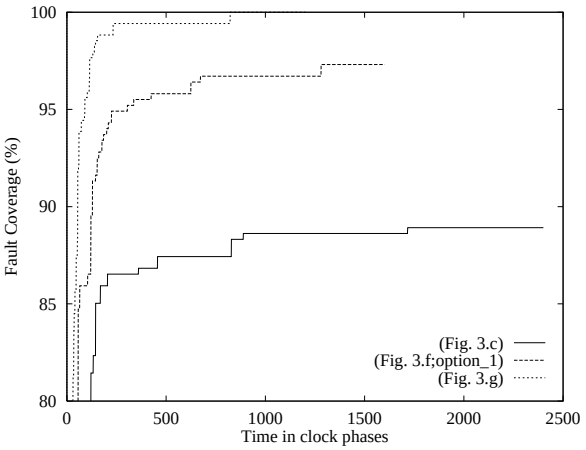

Figure 4: Fault Coverage of the *Polynomial* example for different test schedules corresponding to the MSDGs of Fig. 3(c), (f) and (g)

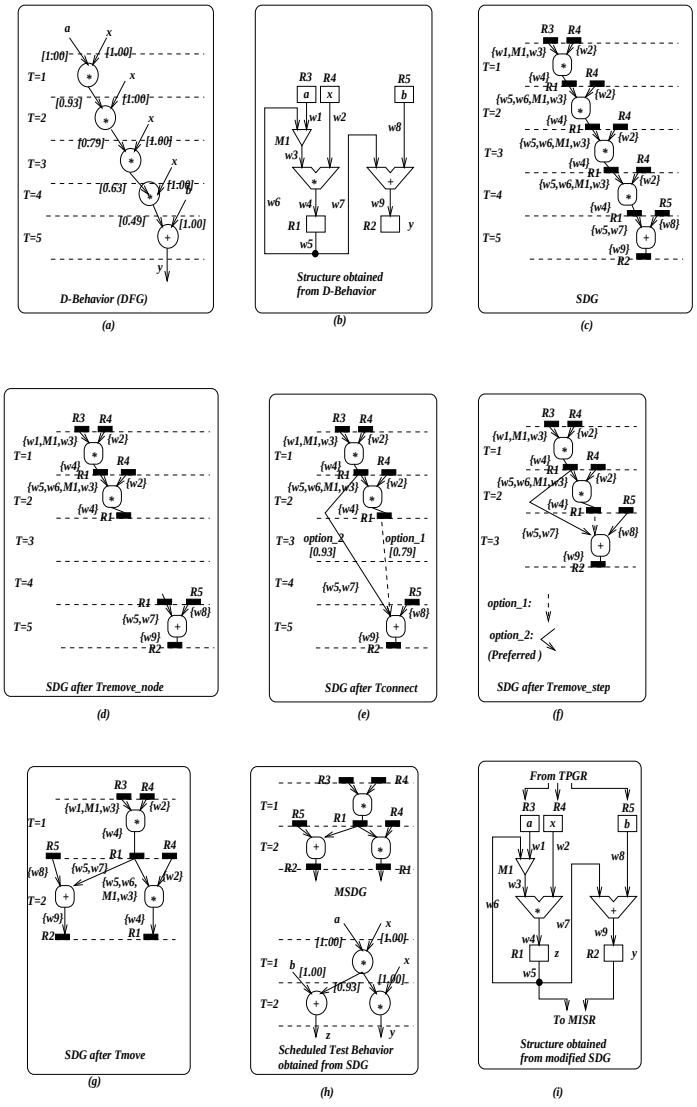

Figure 3: Applying transformations to a small example

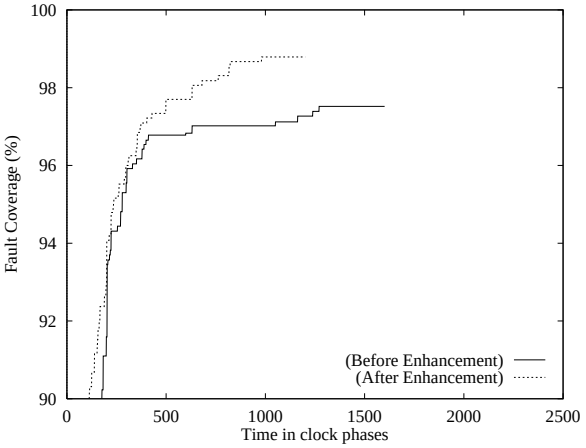

Figure 5: Fault Coverage for the *HAL* example

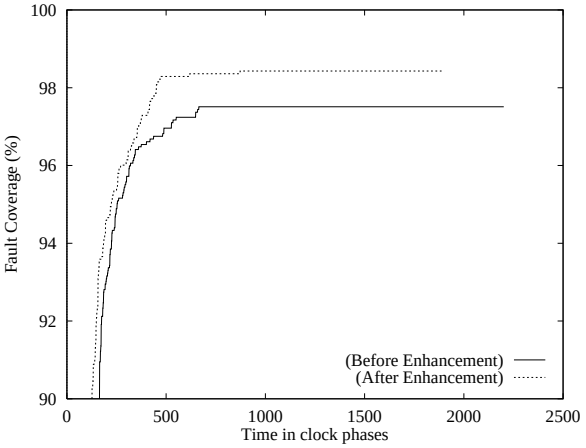

Figure 6: Fault Coverage for the *wave filter* example