Modular Test Generation and Concurrent Transparency-Based Test Translation Using Gate-Level ATPG*

Yiorgos Makris
UCSD - CSE Department
La Jolla, CA 92093

Alex Orailoğlu
UCSD - CSE Department
La Jolla, CA 92093

Praveen Vishakantaiah
Intel Corporation
Hillsboro, OR 97124

Abstract

We introduce a hierarchical test generation methodology for modular designs, employing exclusively gate-level ATPG. Based on the notion of modular transparency, the search space of the design is reduced to alleviate the complexity of gate-level test generation. Although ATPG is applied at the full circuit, faults in each module are targeted individually, while the surrounding modules are replaced by their much simpler, transparency-equivalent logic. As analyzed theoretically and as demonstrated through a set of experimental data, the proposed methodology results in significant test generation speed-up, while preserving comparable fault coverage and vector count to full-circuit gate-level ATPG.

Introduction

Current state-of-the-art in test generation research provides several alternative methodologies towards improving fault coverage or reducing test generation time. Addressing both these constraints however, while using established tool flows to meet time-to-market constraints, has been a challenging and unanswered task. Fault coverage may be boosted through *Modular Decomposition* and *Local Test Generation,* while test generation time may be reduced through *Functional Abstraction* and *High-Level Translation* of locally generated vectors into global design tests. Nevertheless, it is still impractical to employ the complete functionality of the surrounding logic and translate test in a vector-by-vector manner. To expedite this task, a subset of this functionality, defined as *transparency,* is utilized and *symbolic* test translation is performed. Transparency, however, has to preserve adequate test translation capabilities to ensure high global fault coverage. Additionally, utilization of established tool flows requires a mechanism for bridging the high-level transparency definition to gate-level tools.

The methodology proposed in this paper employs modular decomposition for efficient local test generation and functional abstraction for transparency-based local to global test translation. The test translation method is based on a comprehensive transparency definition introduced in (5). Implementation challenges are addressed by identifying modular transparency at the RTL and creating a transparency-equivalent gate-level design view for each module. Modular test generation and concurrent local to global test translation is subsequently performed, using only gate-level ATPG. In the rest of the paper, we summarize related work, introduce the proposed methodology, analyze its effectiveness and support it through experimental results.

Related Work

Several gate-level and hierarchical test generation directions have been investigated to address modern designs. Genetic test generation algorithms, driven by fault simulation are described in (1,11), while new deterministic approaches are discussed in (2). Combined techniques are introduced in (12) and parallel test generation is proposed in (10). Hierarchical test knowledge is extracted in terms of *modes* in (14), which are combined into test translation paths in (15). High-level *architectural information* is used for enhancing hierarchical test generation in (4). Finally, highly translatable local test is generated using commercial ATPG in (13), by incorporating global design *constraints* in the test generation process.

Proposed Methodology

The proposed scheme comprises two phases for each module, as shown in Fig. 1. In the first phase, the recursive design traversal algorithm introduced in (6) searches for RTL test translation paths for each module. The modular transparency behavior on these paths is captured in terms of RTL *transparency channels* (5), requiring an additional RTL tool for performing the local to global test translation. To circumvent this requirement, the second phase generates a simplified gate-level, transparency-equivalent design view for each module. Within this view, the module under test is left intact, while the surrounding modules are replaced by a gate-level implementation of the transparency channels. Gate-level ATPG is applied directly, generating vectors for the faults in the module under test and concurrently translating the vectors through the transparency logic of the surrounding modules.

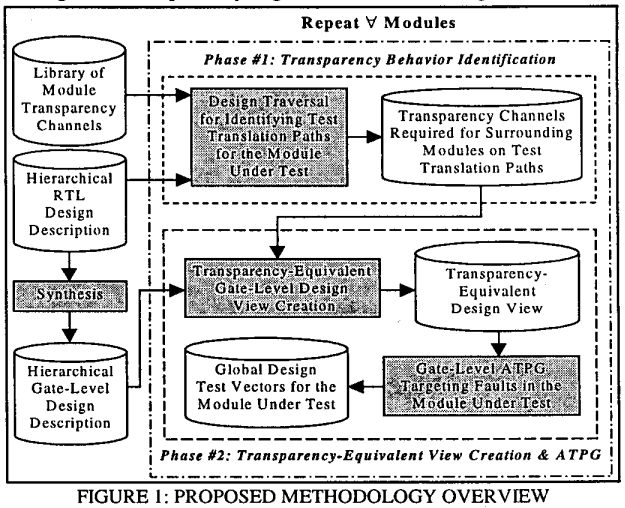

FIGURE 1: PROPOSED METHODOLOGY OVERVIEW

* This work is supported in part through a research grant from Intel Corporation and the University of California MICRO program.

0-7803-5809-0/00/$10.00 © 2000 IEEE IEEE 2000 CUSTOM INTEGRATED CIRCUITS CONFERENCE

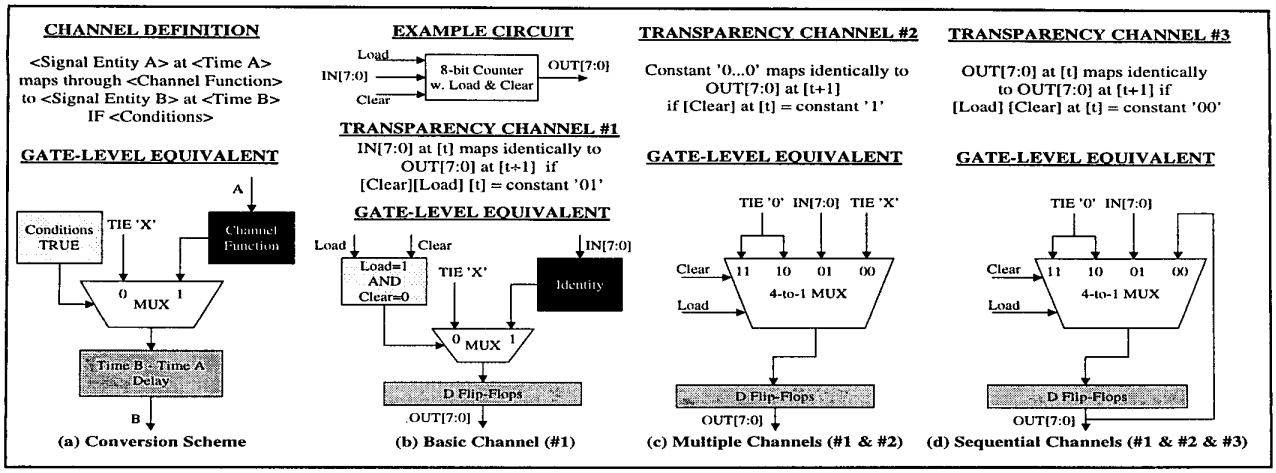

FIGURE 2: GATE-LEVEL TRANSPARENCY-EQUIVALENT VIEW CREATION

Gate-Level Transparency-Equivalent View

In order to create the transparency-equivalent design view, the transparency channels required for each surrounding module need to be converted into gate-level logic. The conversion scheme is depicted in Fig. 2a, where the general channel definition and the corresponding gate-level logic are shown. The underlying assumption supporting this conversion scheme is that the ATPG tool will avoid the Xs during the search. Any behavior other than the transparency channels is modeled as an X through this conversion scheme. Only when the conditions required for the transparency behavior are *TRUE*, a non-X value is achieved. Therefore, the ATPG search is implicitly forced to utilize only transparency behavior by setting the conditions of the MUX to *TRUE*. Sequential transparency is handled by delaying the signal appropriately.

The conversion scheme is also demonstrated on an example circuit, a simple 8-bit counter with load and reset, which comprises three transparency channels. Channel #1 captures the loading capability of the counter, channel #2 captures the clearing capability and channel #3 captures the state holding capability. In Fig. 2b, we demonstrate the conversion of the basic loading channel #1. If only the loading capability of the counter is required, then the counter is replaced by the depicted logic. During test generation, whenever the ATPG tool needs to traverse this logic it will avoid the Xs, therefore forcing the MUX select line to 1 and imposing the desired behavior of loading the counter. When more than one transparency channel is required, the MUX is controlled by a combination of the corresponding channel conditions, as shown in Fig. 2c. In this case, both the loading and the clearing capability of the counter are included in the transparency equivalent design through channels #1 & #2, and it is left to the ATPG tool to decide which one to use each time the substituted logic is traversed. Finally, sequential transparency functions that hold state, such as channel #3, are also supported in the conversion scheme, as shown in Fig. 2d. In this case, all three channels have been included. Based on this methodology, a gate-level transparency-equivalent design is created for each module, wherein the module under test is left intact, while the surrounding modules are converted into their transparency-equivalent views.

Analysis

In this section we discuss the impact of transparency-based test translation on fault coverage and test generation time. Regarding fault coverage, for each fault targeted in a module, full-circuit ATPG searches through the complete functionality of the surrounding modules. Appropriate vector justification and response propagation behavior is found, unless the upper time limit is reached first. In complex designs this happens frequently, resulting in many aborted faults. In the proposed methodology, the search space for each surrounding module is reduced. Inevitably, as the search is performed on a reduced space, some behavior required for test translation may not be present. On the positive side, however, previously aborted faults due to time constraints may become testable because of the reduced search space. In large and complex designs, full circuit ATPG results in many aborted faults that the proposed methodology has an improved chance of covering. Additionally, it has been demonstrated in (7) that transparency channels capture most of the test translation behavior required. Therefore, minimal fault coverage hit, if any at all, can be expected.

In order to analyze the impact of transparency on test generation time, we first make the following observations:

- The time that a test generation tool spends on aborting a fault is longer than the time it spends on proving a fault redundant or testable.
- The time that full circuit ATPG spends on aborting a fault in the original design is equal to the time spent by transparency-based ATPG on aborting a fault in the channel design. This happens because the upper time limit per fault is independent of the design.
- The time that full circuit ATPG spends on proving a fault redundant is on average longer than the time spent by transparency-based ATPG, due to the larger search space.

TABLE I: POSSIBLE TEST GENERATION RESULTS

Full Circuit ATPG Reports Fault As:	Fault in Original Circuit Is:	Transparency-Based ATPG Reports Fault As:
Testable	*Testable*	*Testable*
		Redundant
Redundant	*Redundant*	*Redundant*
Aborted	*Testable*	*Testable*
		Redundant
		Aborted
	Redundant	*Redundant*
		Aborted

- Similarly, the time that full circuit ATPG spends on proving a fault testable is on average longer than the time spent by transparency-based ATPG.

The above observations are used to compare the time spent on a fault by full circuit and transparency-based test generation. Table I summarizes all the possible combinations of test generation outcomes of the two alternatives. Applying these observations on each row of Table I establishes that for the complete set of faults, transparency-based ATPG will be significantly faster than full circuit ATPG.

Experimental Results

The proposed methodology is evaluated by means of two distinct experiments. The first experiment examines the adequacy of the transparency-equivalent design views for test translation, while the second compares the proposed test generation scheme to full circuit ATPG. The experiments are performed on two RTL designs, a 3-module control circuit (MTC100) originally described in (14) and an 8-bit sign-magnitude binary multiplier comprising eight modules and a controller, as described in (3). In this section we explain the setup for each experiment and we discuss the results.

Exhaustive vs. Transparency-Based Translation: The setup for the first experiment is depicted in Fig. 3. For each design module *k*, we perform two ATPG runs using HITEC (8). The first run is performed on the original design, while the second run is performed on the transparency-equivalent design view for module *k*. The vectors obtained from each run are fault simulated on the original design using PROOFS (9). Both during ATPG and fault simulation, only faults in module *k* are targeted. Test translation efficiency is evaluated based on fault coverage, test generation time and vector count comparison of the two ATPG runs. Tables II and III show the results for the two example circuits respectively. Results are reported for each module and accumulated for all modules in the bottom column of each table. As explained in the previous section, testable faults in the original design may be reported as redundant in the transparency-equivalent view. On the positive side, previously aborted faults may become testable, due to reduced search space. The results show a significant test generation time speedup in the proposed scheme, while coverage remains comparable, demonstrating the adequacy of the transparency-equivalent view for test translation.

Full Circuit vs. Transparency-Based ATPG: The setup for the second experiment is shown in Fig. 4. For the purpose of full circuit ATPG, HITEC (8) provides fault coverage, test generation time and vector count. For the purpose of transparency-based ATPG, HITEC (8) is applied on the transparency-equivalent design view of the first module, targeting only faults in this module. The obtained vectors are fault simulated on the original design for all design faults. The process is repeated for each module, this time targeting only faults that have not been covered by vectors obtained for previous modules. Fault coverage, test generation time, and vector count are thus obtained for each module and accumulated for the complete design. The results for the two example circuits are shown in Tables IV and V, respectively, demonstrating a significant test generation time reduction in the proposed scheme, as compared to full circuit ATPG. Interestingly, the speed-up increases with the size of the circuit, since full circuit ATPG has an increasingly larger search space to deal with than the proposed methodology. Fault coverage not only is comparable but furthermore *improves* in the case of the binary multiplier. This is due to aborted faults in the full chip ATPG run for which the proposed method is able to generate tests. Vector count increases slightly in the first design but decreases in the second, while still providing improved fault coverage, thus demonstrating the superiority of the proposed method over full-circuit ATPG.

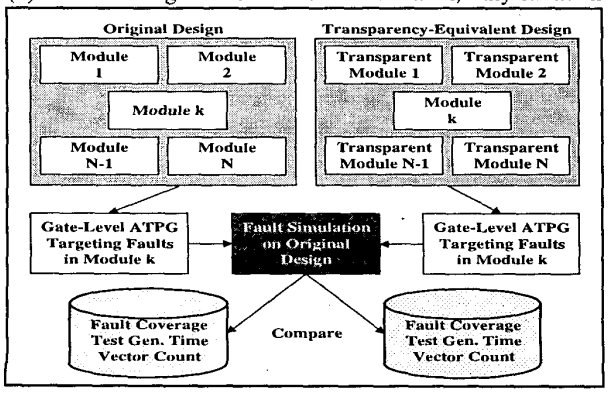

FIGURE 3: SETUP FOR EXPERIMENT #1

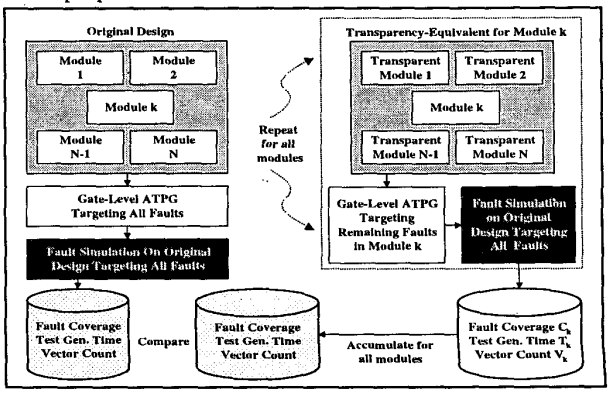

FIGURE 4: SETUP FOR EXPERIMENT #2

TABLE II: RESULTS OF EXPERIMENT #1 ON MTC100

Module Name	Total Faults	Covered		Aborted		Redundant		Vector Count		T. G. Time (sec)	
		Orig.	Transp.	Orig.	Transp.	Orig.	Transp.	Orig.	Transp.	Orig.	Transp.
Counter	258	256	258	2	0	0	0	103	113	0.483	0.133
Adder	657	561	561	6	6	0	0	77	109	1.133	0.300
Register	230	213	209	17	21	0	0	57	64	1.867	0.067
Total	1055	1030	1028	25	27	0	0	237	296	3.483	0.500

TABLE III: RESULTS OF EXPERIMENT #1 ON MULTIPLIER

Module Name	Total Faults	Covered		Aborted		Redundant		Vector Count		T. G. Time (sec)	
		Orig.	Transp.	Orig.	Transp.	Orig.	Transp.	Orig.	Transp.	Orig.	Transp.
A0	21	19	18	2	0	0	3	62	62	0.333	0.133
Adder	234	197	212	37	22	0	0	80	92	9.417	4.017
Areg	143	141	139	2	1	0	3	124	144	1.833	0.350
Array	32	32	31	0	0	0	1	10	33	0.300	0.033
Mreg	91	90	90	0	0	1	1	63	75	1.100	0.083
Qreg	153	153	153	0	0	0	0	69	84	0.233	0.067
Sign	10	10	6	0	0	0	4	72	36	0.167	0.067
Mux	8	7	6	1	0	0	2	36	36	0.467	0.083
Control	142	127	127	6	6	9	9	88	88	0.950	0.717
Total	834	776	782	48	29	10	23	606	650	14.800	5.550

TABLE IV: RESULTS OF EXPERIMENT #2 ON MTC100

Module Name	Covered Faults	Vector Count	T. G. Time (sec)
Full Circuit	1034	146	3.383
Adder	983	109	0.300
Counter	49	64	0.067
Register	2	7	0.033
Total	1034	180	0.400

TABLE V: RESULTS OF EXPERIMENT #2 ON MULTIPLIER

Module Name	Covered Faults	Vector Count	T. G. Time (sec)
Full Circuit	760	191	11.583
Areg	777	144	0.350
Adder	13	16	3.633
Control	0	0	0.533
A0	0	0	0.033
Mreg	0	0	0.017
Total	790	160	4.566

Conclusions

We propose a methodology that reduces the complexity of hierarchical test generation, while employing only gate-level tools. The concept of modular transparency is utilized for identifying test translation behavior and a gate-level transparency-equivalent design view is generated for each module in the design. Subsequently, gate-level ATPG tools are applied on the simplified transparency-equivalent design views, generating and concurrently translating test for each module. As discussed theoretically and as confirmed by the experimental results, the transparency equivalent design views reduce substantially the test generation search space, while preserving adequately the required test translation behavior. As a result, an efficient gate-level hierarchical test generation methodology is devised that significantly reduces test generation time while preserving fault coverage and vector count comparable to full circuit gate-level ATPG.

References

(1) F. Corno, P. Prinetto, M. Rebaudengo, M. Sonza Reorda, "GATTO: A Genetic Algorithm for Automatic Test Pattern Generation for Large Synchronous Sequential Circuits", *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 15, no. 8, pp. 991-1000, 1996.

(2) I. Hamzaoglu, J. H. Patel, "New Techniques for Deterministic Test Pattern Generation", *Proceedings of the VLSI Test Symposium*, pp. 446-452, 1998.

(3) J. P. Hayes, *Computer Architecture and Organization*, McGraw-Hill, 3rd Edition, 1998.

(4) J. Lee, J. H. Patel, "Hierarchical Test Generation under Architectural Level Functional Constraints", *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 15, no. 9, pp. 1144-1151, 1997.

(5) Y. Makris, A. Orailoğlu, "RTL Test Justification and Propagation Analysis for Modular Designs", *Journal of Electronic Testing: Theory & Applications*, Kluwer Academic Publishers, vol. 13, no. 2, pp. 105-120, 1998.

(6) Y. Makris, A. Orailoğlu, "DFT Guidance Through RTL Test Justification and Propagation Analysis", *Proceedings of the International Test Conference*, pp. 668-677, 1998.

(7) Y. Makris, J. Collins, A. Orailoğlu, P. Vishakantaiah, "TRANSPARENT: A System for RTL Testability Analysis, DFT Guidance and Hierarchical Test Generation", *Proceedings of the Custom Integrated Circuits Conference*, pp. 159-162, 1999.

(8) T. Niermann, J. H. Patel, "HITEC: A Test Generation Package for Sequential Circuits", *Proceedings of the European Conference on Design Automation*, pp. 214-218, 1992.

(9) T. Niermann, W. T. Cheng, J.H. Patel, "PROOFS: A Fast, Memory Efficient Sequential Circuit Fault Simulator", *Proceedings of the Design Automation Conference*, pp. 535-540, 1990.

(10) B. Ramkumar, P. Banerjee, "Portable Parallel Test Generation for Sequential Circuits", *Proceedings of the International Conference on Computer-Aided Design*, pp. 220-223, 1992.

(11) E. M. Rudnick, J. H. Patel, G. S. Greenstein, T. M. Niermann, "A Genetic Algorithm Framework for Test Generation", *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 16, no. 9, pp. 1034-1044, 1997.

(12) D. G. Saab, Y. G. Saab, J. A. Abraham, "Iterative (Simulation-Based Genetics + Deterministic Techniques) = Complete ATPG", *Proceedings of the International Conference on Computer-Aided Design*, pp. 40-43, 1994.

(13) R. S. Tupuri, J. A. Abraham, "A Novel Test Generation Method for Processors using Commercial ATPG", *Proceedings of the International Test Conference*, pp. 743-752, 1997.

(14) P. Vishakantaiah, J. A. Abraham, M. S. Abadir, "Automatic Test Knowledge Extraction From VHDL (ATKET)", *Proceedings of the Design Automation Conference*, pp. 273-278, 1992.

(15) P. Vishakantaiah, J. A. Abraham, D. G. Saab, "CHEETA: Composition of Hierarchical Sequential Tests using ATKET", *Proceedings of the International Test Conference*, pp. 606-615, 1993.