An approach for verification assertions reuse 2 in RTL test pattern generation

Maksim Jenihhin[1], Jaan Raik[1], Raimund Ubar[1],
Taavi Viilukas[1], Hideo Fujiwara[2]

(1. Department of Computer Engineering, Tallinn University of Technology, Tallinn, Estonia;

2. Graduate School of Information Science, Nara Institute of Science and Technology, Nara, Japan)

Abstract: Assertions are used in functional verification of design to detect design errors. In this paper we propose an approach for their reuse in manufacturing test pattern generation at Register-Transfer Level (RTL) for non-scan designs. The approach provides search-space reduction for sequential ATPG therefore potentially speeding up the test generation process and increasing the fault coverage. A discussed case-study demonstrates the feasibility and effectiveness of the proposed idea.

Key words: RTL; ATPG; assertions; non-scan designs

CLC number: TP302.8 **Document code**: A **Article ID**: 1000-5137(2010)05-0441-07

1 Introduction

Test pattern generation for today's sequential circuits is lacking satisfactory methods and remains to be a challenge for both industry and academia. One of the wide spread solutions used by the community at present is substitution of the hard test pattern generation task by theoretically much simpler approach relying on scan-paths together with combinational Test Pattern Generation (TPG). However, the scan-path methods have their shortcomings including increased area, delay and consumed power. It also causes targeting of non-functional failure modes, which results in over-testing and yield loss. In the rest of the paper we will consider circuits under test without scan chains or other DFT (design for testability) solutions.

In order to cope with the non-scan TPG problem a number of approaches have been proposed. The ones targeting deterministic TPG at the gate level[7] cannot efficiently handle sequential designs starting from a couple of thousands of gates. The simulation-based approaches[8] in turn cannot guarantee detection of hard-to-test faults. The fundamental shortcoming of the functional test generation approaches[9] that rely on functional fault models is that they do not offer full structural level fault coverage. Hierarchical and RTL test pattern generation has been proposed[10] as a promising alternative to target complex sequential circuits. The published works include implementing assignment decision diagram models combined with SAT methods to address register-transfer level test pattern generation[11]. In [1] and [2] we have proposed a hierarchical constraint-based TPG for

Received date: 2010-09-25

Foundation item: This work has been supported in part by Estonian Science Foundation through grants 8478 and 7068, European Commission projects FP7-2009-IST-4-248613 DIAMOND and FP7-REGPOT-2008-1 CREDES, by Research Centre CEBE funded by EU Structural Funds.

Biography: Maksim Jenihhin, Doctor, Department of Computer Engineering, Tallinn University of Technology, email: maksim@ati.ttu.ee.

RTL designs. Its advantages as well as some limitations will be discussed in more details in the next section.

In this paper we propose to have a broader look at the discussed above problem of TPG for manufacturing test. The preceding phases of an ASIC (application specific integrated circuit) development flow normally include the design phase which is tightly coherent with the functional verification process targeted at design errors. The main goal of the functional verification is to ensure the functionality of the design implementation (normally expressed by means of hardware description languages i. e. HDLs) corresponds to the requirements of the specification prior the synthesis phase. The verification process can rely on both formal and simulation-based approaches. The verification is a hard task by itself and intensive research goes in this area as well. One of the efficient strategies used in verification is application of assertions[3], which are pieces of a design's explicitly specified behavior and aimed at design hard to verify parts. The recent emergence and success of such assertion specification languages as PSL (Property Specification Language)[4] and SystemVerilog[5] is an important step in assertion-based verification methodology development. The assertions can be used in both formal and simulation-based verification approaches, however normally they are cleaned out from the HDL code once the verification process is finished and the design is sent for synthesis.

The approach we propose in this paper considers reuse of the information functional verification assertions contain for TPG targeted at structural manufacturing test. One of the important observations here is that normally the assertions are written by the design engineer who has a deep understanding of the design's functionality.

In [6] we have discussed the ideas for verification assertions reuse directions very generally. In [12] and [13] the authors address hardware checkers generation from assertions targeted to aid manufacturing testing.

As opposed to the mentioned approaches we consider assertions as additional information for deterministic TPG targeting RTL non-scan designs.

The rest of the paper is organized as follows. Section 2 describes the existing hierarchical constraint-based TPG for RTL designs called DECIDER. Section 3 introduces the proposed approach for verification assertions reuse for RTL TPG. A case-study based on ITC'99 benchmark circuit *b*02 is presented here for explanation of the approach. Section 4 concludes the paper.

2 RT-level test pattern generator decider

In [1] and [2] we have proposed a hierarchical test generation approach for non-scan designs at RTL. The high-level symbolic path activation, described in this section is a complete algorithm, i. e. if transparent paths for fault effect propagation and value justification exist, they will be activated. The algorithm has been implemented as a systematic search and therefore an inconsistency in any stage causes a backtrack and a return to the last decision. However, due to the NP-complete nature of the problem, in some cases, the search must be terminated after a certain maximal number of solutions have been tried.

The approach has two main phases. During the first phase, high-level test path activation, an untested module is selected and for this module propagation and justification is performed. In addition, constraints for the test path are extracted. The goal of the second phase is to satisfy the constraints by using a constraint solver and to compile the test patterns by assigning the values obtained by the constraint solver to the primary input signals. For this purpose an open source constraint solver ECLiPSe[14] is used.

The high-level test generation constraints are divided into three categories. These are path activation constraints, transformation constraints and propagation constraints. Path activation constraints correspond to the logic conditions in the control flow graph that have to be satisfied in order to perform propagation and value justification through the circuit. Transformation constraints, in turn, reflect the value changes along the paths from the inputs of the high-level Module Under Test (MUT) to the primary inputs of the whole circuit. These constraints are needed in order to derive the local test patterns for the module under test. Propagation constraints show how the value propagated from the output of the MUT to a primary output is depending on the values of the signals in the system. The main idea here is to guarantee that fault signals will not be masked when propagated. All the above categories of constraints are represented by common data structures and manipulated by common procedures for creation, update, modeling and simulation.

In our previous works we have proven the DECIDER to be an efficient tool for RTL circuits TPG. Table 1[19] presents the characteristics of the example circuits used in test pattern generation experiments in this paper. The following benchmarks were included to the test experiment: a Greatest Common Divisor (*gcd*16), an 8-bit multiplier (*mult*8 ×8), an Elliptic Filter (*ellipf*), an ALU based processor (*risc*) and a Differential Equation (*diffeq*). The VHDL versions of *gcd*16 and *diffeq* were obtained from high-level synthesis benchmark suites[16,17] and the designs of mult 8 ×8 and *risc* from functional test generation (FUTEG) benchmarks[18]. The second column"# faults" shows the number of single stuck-at faults in the circuits, the third column "# FSM states" shows the number of states in the control part FSM, and the columns "PI bits" and "PO bits" present the number of primary input and primary output bits, respectively. Finally, the 6th, 7th and 8th columns show the number of registers, multiplexers and functional units respectively.

In Table 2[19], comparison of test generation results of three sequential ATPG tools on the hierarchical benchmark designs are presented. These include a gate-level deterministic ATPG HITEC[7], a genetic algorithm based GATEST[8], and DECIDER[19]. Columns "F. C./%" give the single stuck-at fault coverages of the test patterns generated measured by the fault simulator from TURBO TESTER system[15], created at Tallinn University of Technology. Columns"time/s" stand for test generation run-times achieved on a 366 MHz SUN UltraSPARC 60 server with 512MB RAM under SOLARIS 2.8 operating system. The results show that DECIDER is very efficient for testing sequential designs. It achieves in average 2.5% higher fault coverage than the genetic tool GATEST on the given benchmark set.

Table 1 Characteristics of the benchmark circuits

circuit	# faults	# FSM states	PI bits	PO bits	# of reg.	# of mux	# of FU
gcd16	1754	8	33	16	3	4	3
mult8x8	2036	8	17	16	7	4	9
ellipf	5388	28	130	113	17	7	3
risc	6434	4	26	16	8	4	4
diffeq	10008	6	81	48	7	9	5

Table 2 Comparison of sequential circuit test generation tools

circuit	HITEC		GATEST		DECIDER	
	F. C./%	time/s	F. C./%	time/s	F. C./%	time/s
gcd16	59.11	365	86.13	190.7	**90.95**	677.4
mult8x8	65.9	1243	69.2	821.6	**74.7**	93.7
ellipf	87.9	2090	94.7	6229	**95.04**	1258.9
risc	52.8	49,020	96.0	2459	**96.5**	150.5
diffeq	96.2	13,320	96.40	3000	**97.09**	453.7
aver. F. C.:	72.4		88.4		**90.9**	

3 Application of assertions for TPG

DECIDER relies on HLDD representations[19] of the design under test in order to generate the test patterns. The tool is capable of modeling FSMs, however, it is unable to target nodes in the FSM itself. This is due to the fact that the concept of testing FSMs is very different from datapath testing. When targeting datapaths, then the steps of fault manifestation, fault effect propagation and value justification are performed. Values are propagated through the datapath and FSM is taken into account only to keep track of the control state sequence.

However, when targeting FSMs and control dominated circuits then the approach differs. Here we need to:

Step A: activate a state sequence to the control state (or state transition) under test.

Step B: differentiate the fault-free and faulty control states (or state transition).

Step C: activate a sequence propagating this difference to observable outputs.

Consider the following motivational example based on the ITC99 benchmark circuit *b*02[21] presented in Figure 1 shows the state diagram of the circuit.

The circuit has one input signal called input, one output signal called output, and one internal variable state. In the state diagram, the diagram nodes are labeled by FSM states $\{A, B, C, D, E, F, G\}$ the edges are labeled by the values of inputs, which activate the corresponding transition and the output values at that transition. The input and output values are separated by a slash symbol. In the HLDD presented in Figure 4 the non-terminal nodes are labeled by inputs and current state and the terminal nodes are labeled by output and next state values, respectively. The HLDD computes values to a vector of design variables {state, output} during each clock cycle.

The fault models targeted during the test generation process by DECIDER for both FSM and datapath are expressed[19] using HLDDs.

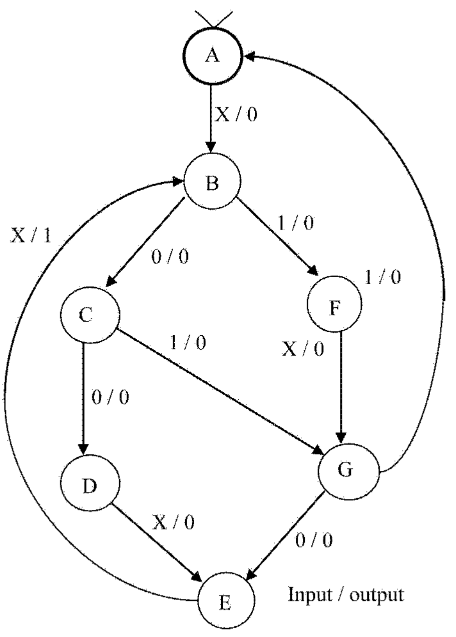


Figure 1 The FSM of the case-study circuit *b*02

Consider an incomplete set of verification assertions written in PSL language:

*p*1: assert always ({ (state = A) ; [* 3] ; ! input; } | = > { output }) ;

*p*2: assert always (input and ! (state = D) → next ! output) ;

These two assertions represent checks for functional correctness of the FSM implementing the *b*02 design. The first assertion *p*1 states that if we have the following sequence of signal values: first we are in state *A*, and then after a three don't-care clock cycles we have input set to 0 then on the next clock cycle (| = > is a non-overlapping implication operator for sequences in PSL) output will be set to 1. The second assertion *p*2 is interpreted as follows. If input is 1 and we are not in state *D* then at the next clock cycle output must be 0.

In a real design flow the verification engineer writes a longer set of assertions that represents properties specifying the behavior of the circuit. Such information, although created for verification purposes, could be used by the automated test generation algorithm because it contains some high-level knowledge about the functionality of the design.

For example, property *p*1 can be beneficial in activating the test sequence for value justification (Step A of the FSM test generation, mentioned above). Assume that we need to justify state E, which is the only state where output is one, by backtracing a state sequence to the initial state A (see Figure 1). The information that is transferred to the ATPG by $p1$ is that when we justify, it is necessary to set input to 0 after a three arbitrary values to reach E from A. Therefore, the justification sequence is easily derived just by moving to A, holding input equal to 0 and waiting for 4 clock-cycles. Unnecessary backtracks and entering of loops during the systematic search will be avoided. Similarly, the same assertion could be applied in propagation to state E from the initial sate A of the FSM (Step C of FSM test generation).

Property $p2$ may be utilized in distinguishing the fault-free and faulty control states (Step B). For example, if we are in state D then we need to set input to 1 in order to distinguish it from other states.

In a similar manner the information from verification assertions can be reused for TPG targeted at datapath. Generally assertions consist of two parts: precondition and implication separated by the one of the implication operators (e. g. $\rightarrow$). Let's denote the set of signals in the precondition part by S^P and the set of signals in the implication part by S^I.

Let's consider a circuit under test containing two modules (Figure 2). And an abstract assertion W which both S^P and S^I are some of the signals crossed by the curved line in Figure 2.

W: *fPrecondition* (SP) $\rightarrow$ *fImplication* (SI);

Then for a fault F_1 in Module 1 both S^P and S^I can be used as a monitoring constraint, which allows to reduce the propagation time (Figure 3a) required for Step C of FSM test generation flow. In case of a fault F_2 in Module 2 the signals set S^P can be controlled depending on the monitoring results of S^I and thus can be used to reduce the justification time (Figure 3b) required for Step A.

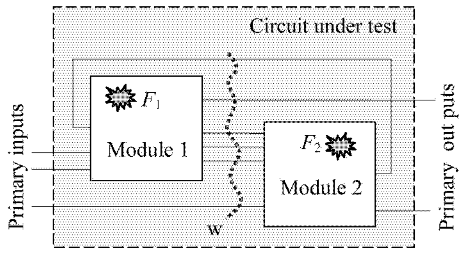
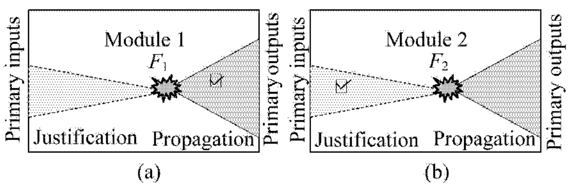


Figure 2　A circuit under test with two modules



(a) fault propagation (b) fault justification

Figure 3　Assertion applicability for TPG

The knowledge from assertions may be forwarded to the ATPG algorithm in the form of implications, similar to combinational gate-level ATPG algorithms taking advantage of implications and learning[23-25]. In order to allow the transfer of knowledge from verification assertions into the ATPG algorithm, both, representation of assertions and derivation of implications from them have to be formalized.

In Section 2 we have discussed constraints for test generation that are derived automatically from the circuit structure. A possible approach for formalization of the assertion information for TPG is their use to provide for additional constraints. For example, the assertions useful for Step A of FSM test generation flow can be used to create additional constraints for the DECIDER's path activation constraints and the ones useful for Step C - for propagation constraints correspondingly. The constraints from assertions allow reducing the number of backtracks while the set of main constraints is being solved and thus can speed-up test generation process and increase the fault coverage.

Usefulness and applicability of the assertion for TPG can be influenced by particular temporal relationships of the expressions involved in the verification assertions (e. g. the ones set by the PSL operator eventually) and complicated correlation of the functionality specified by an assertion to the circuit's structure. Therefore an approach for proper analysis of assertion applicability for manufacturing TPG is required.

4 Conclusions and future work

In the ASIC development flow assertions are used in functional verification of design to detect design errors. This paper has proposed an approach for the assertions reuse in manufacturing test pattern generation at RTL for non-DFT designs. The proposed approach provides for fault coverage increase and speed-up of test generation process. The advantages are achieved by reducing the number of backtracks during the fault justification and propagation procedures of TPG. The discussed case-study with ITC'99 benchmarks family circuit *b*02 demonstrates the feasibility and effectiveness of the proposed idea.

In the future work we aim to formalize the approach for additional constraints creation from the appropriate verification assertions. The other important step for the methodology we would like to address is a proper analysis of the complex temporal verification assertions for their applicability in the proposed approach.

Acknowledgment

This work has been supported in part by Estonian Science Foundation through grants 8478 and 7068, by European Commission projects FP7-2009-IST-4-248613 DIAMOND and FP7-REGPOT-2008-1 CREDES, by Research Centre CEBE funded by EU Structural Funds.

References:

[1] RAIK J, UBAR R. Fast test pattern generation for sequential circuits using decision diagram representations[J]. JETTA, Kluwer, 16(3), 2000.

[2] VIILUKAS T, RAIK J, JENIHHIN M, et al. Constraint-based test pattern generation at the register-transfer level[C]. Proc of IEEE DDECS, 2010.

[3] FOSTER H D, KROLNIK A C. Creating Assertion-Based IP[M]. New York: Springer, 2008.

[4] IEEE-Commission. IEEE Std 1850-2005/2010 IEEE standard for Property Specification Language (PSL)[S]. 2010.

[5] IEEE Computer Society. IEEE Std 1800-2005/2009 IEEE standard for system verilog-unified hardware design, specification, and verification language[S]. 2009.

[6] JENIHHIN M, RAIK J, UBAR R, et al. On reusability of verification assertions for testing[C]. Proc of IEEE BEC'08, Tallinn, Estonia, 2008: 151-154.

[7] NIERMANN T M, PATEL J H. HITEC: A test generation package for sequential circuits[C]. Proc of EDAC, 1991: 214 - 218.

[8] RUDNICK E M, et al. Sequential circuit test generation in a genetic algorithm framework[C]. Proc of DAC, 1994: 698 - 704.

[9] BRAHME D, ABRAHAM J A. Functional testing of micro-processors[J]. IEEE Trans Comput, 1984, 33(6): 475 - 485.

[10] MURRAY B T, HAYER J P. Hierarchical test generation using precomputed tests for modules[J]. Computer-Aided Disign of Integrated Circuits and System, IEEE, 1988, 9(6): 221 - 229.

[11] FUJIWARA H, OOI CY, SHIMIZU Y. Enhancement of test environment generation for assignment decision diagrams[C]. WRTLT, 2008.

[12] KAKOEE M R, RIAZATI M, MOHAMMADI S. Enhancing the testability of RTL designs using efficiently synthesized assertions[C]. Proc of ISQED, 2008: 230 - 235.

[13] BOULE M, CHENARD J S, ZILIC Z. Assertion checkers in verification, silicon debug and in-field diagnosis[C]. Proc of ISQED, 2007: 613 - 620.

[14] ECLiPSe Constraint Programming System[Z/OL]. http://eclipseclp. org/.

[15] Turbo Tester Tools[Z/OL]. http://www. pld. ttu. ee/tt.

[16] HLSynth92 benchmarks[Z/OL]. http://www. cbl. ncsu. edu/pub/Benchmark_dirs/HLSynth92/.

[17] HLSynth95 benchmarks[Z/OL]. http://www. cbl. ncsu. edu/pub/Benchmark_dirs/HLSynth95/.

[18] GRAMATOVA E, GULBINS M, MARZOUKI M, et al. FUTEG Benchmarks. Technical Report of project COPERNICUS, JEP 9624 FUTEG[R]. 1995.

[19] RAIK J, UBAR R, VIILUKAS T, et al. Mixed hierarchical-functional fault models for targeting sequential cores[J]. Elsevier Journal of Systems Architecture Elsevier, 2008, 54(3 - 4): 465 - 477.

[20] UBAR R, RAIK J. Hierarchical test generation for complex digital systems with control and data processing parts[J]. Test, Assembly and Packaging, SEMICON, Singapur, 1999(3 - 6): 43 - 52.

[21] CORNO F, REORDA M S, SQUILLERO G. RT-level ITC'99 benchmarks and first ATPG results[J]. Journal Design & Test of Computers, IEEE, 2000, 17(3): 44 - 53.

[22] POMERANZ I, REDDY S M. Application of homing sequences to synchronous sequential circuit testing[J]. IEEE Trans Comput, 1994, 43(5): 569 - 580.

[23] TAFERTSHOFER P, GANZ A, HENFTLING M. A SAT-based implication engine for efficient ATPG, equivalence checking, and optimization of netlists[C]. Proc of Int Conf CAD, 1997.

[24] MUKHERJEE R, JAIN J, FUJITA M, et al. On more efficient combinational ATPG using functional learning[C]. Proc of Int Conference on VLSI Design: VLSI in Mobile Communication, 1996.

[25] BOMMU S, CHANDRASEKAR K, KUNDU R, et al. CONCAT: CONflict driven learning in ATPG for industrial designs [C]. Proc of International Test Conference, 2008.

一种在 RTL 测试模式生成中验证断言再用的方法

马克西姆 捷尼赫尼[1], 捷安 瑞克[1], 莱穆德 俄巴尔[1], 塔维 维卢卡斯[1], 藤原秀雄[2]

(1. 塔林工学院 计算机工程系, 塔林, 爱沙尼亚; 2. 奈良科技研究所 信息科学研究生院, 奈良, 日本)

摘　要: 在对设计的功能验证中, 断言常被用于检测设计错误. 针对制造业的测试模式生成, 提出了在寄存器传输层(RTL)用于无扫描设计的断言再用方法. 这种方法减少了顺序自动测试码生成程序(ATPG)的搜索空间, 因而能加快测试生成过程, 增加故障覆盖率. 通过实例分析, 证明了该方法的可行性和效果.

关键词: 寄存器传输层; 自动测试码生成程序; 断言; 无扫描设计

(特约编辑: 高建华)