A Synthesis-Agnostic Behavioral Fault Model for High Gate-Level Fault Coverage

Anton Karputkin Jaan Raik
Department of Computer Engineering
Tallinn University of Technology, Estonia
{anton.karputkin | jaan.raik}@ati.ttu.ee

***Abstract*—Early design space exploration is a practice for avoiding issues that manifest themselves at late design phases. Nevertheless, the test development has traditionally been postponed to the final stages of the design process. At the same time, more and more IP designs are sold at the RTL, where details of exact gate-level implementation are unknown. While a range of RTL ATPG methods has been developed over the past decades, the fault models are too inaccurate in order to guarantee full coverage for the gate-level faults. This paper fills the gap by proposing a synthesis-agnostic ATPG based on extending behavioral fault models in order to allow targeting stuck-at faults in the gate-level implementations of RTL designs regardless of the synthesis decisions made. Moreover, the approach does not require adding scan paths and therefore the obtained test sequences serve as at-speed, functional mode tests. Experiments on a set of benchmarks and an industrial design show that the proposed fault models are superior to the previous approaches in terms of stuck-at fault coverage. Comparison with a state-of-the-art gate-level sequential ATPG show higher or equal coverage for the proposed technique achieved at shorter runtimes.**

I. Introduction

Nowadays, the test development is still postponed to the final stages of the design process. On the other hand, issues detected at such late phases would cause significant increase in time-to-market. At the same time, more and more Intellectual Property (IP) designs are sold at the Register-Transfer Level (RTL). It is the task of the customer to synthesize such IP according to selected technology and develop the test. However, design productivity would be greatly improved if a synthesis-agnostic reusable test was supplied with the IP.

In order to generate tests at the RTL, sequential Automated Test Pattern Generation (ATPG) should be applied. There is a wide range of approaches developed for sequential ATPG over the recent decades. These include gate-level deterministic [1,2] as well as simulation-based approaches, the latter being mainly based on genetic algorithms [3,4] and spectral methods [5]. However, the gate-level approaches suffer from the fact that the test generation efficiency in terms of fault coverage and run time does not scale with the circuit size.

In order to cope with the underlying complexity, high-level ATPG has been proposed [6]. Test generation algorithms applied at higher levels are mainly based on formal techniques. Recently, there has been a breakthrough in SMT-solvers that allow creating more and more powerful formal reasoning engines. First SMT-based ATPG approaches have been already developed [7].

However, as we show in this paper, the previous high-level fault models were not accurate enough in order to fully cover the gate-level faults. The existing Register-Transfer Level (RTL) fault models [8] rely on unrealistic assumptions of the coding style, where the data-path is expected to be represented structurally with functional units, multiplexers and registers clearly separated into modules. A wide-spread design entry is the so called High-Level State Machine (HLSM), where the data variables and control states are provided and the operator mapping, multiplexers and register control signals are missing. The behavioral fault models that target this abstraction level lack accuracy in terms of matching the structural faults [9] due to the fact that several synthesis decisions are not known.

In order to overcome this limitation more accurate behavioral fault models are required to be able to aim high structural fault coverage in designs with realistic coding styles. This paper exceeds current state-of-the-art in high-level ATPG in this aspect by proposing:

- a new set of behavioral fault models that allows targeting designs with complex control part structures and counters as well as detection of stuck-at faults (SAF), which are testable by a gate-level ATPG in the ITC99 gate-level benchmarks. Faults are presented in a form of bit-vector constraints that makes them directly applicable into SMT-based frameworks.
- an experimental study of previously developed RTL fault models in order to assess their validity in a behavioral set up, which shows that the fault coverage is always less than the one by the proposed approach.

The paper is organized as follows. Section II discusses representing HLSM descriptions by FSMD model. Section III proposes the set of fault models for HLSM. Finally, experimental results and conclusions are provided.

II. FSMD Model

While researchers are making constant attempts to increase the abstraction level of hardware descriptions, RTL is going to remain the dominant design entry for the following years [18]. More precisely, a *High-Level State Machine (HLSM)* [19] is often implemented. Improving fault coverage at this level is the goal of current paper. More precisely, we assume that the design is cycle-accurate and we have mapping for the registers and inputs/outputs. However, other structural information, such as multiplexer instantiations, and operations mapping to functional units, is missing.

In this work we utilize the following notation: the set of all registers except the state register is denoted by R; for each register $r \in R$, A_r stands for the set of right-hand-sides of assignments, where r is the left-hand side, appended with a fictitious assignment $r := r$, which takes place whenever r retains its current value for the next clock cycle.

We assume that designs described as HLSM after the synthesis will possess the structure known as *Finite State*

978-3-9815370-6-2/DATE16/ ©2016 EDAA

Machine with Data-path (*FSMD*) [10] which implements a digital system as the interaction of a *data-path* and a *controller*. Fig. 1 shows basic building blocks of a typical FSMD and their possible relation to each other: the data-path consumes inputs, stores them in registers and/or memory, applies arithmetical or logical operations and produces output signals. This process is managed by the control unit that, given the state register value, inputs and signals computed by functional units, produces selection values that govern what to write to storage elements for the next time step and transfers to the next state.

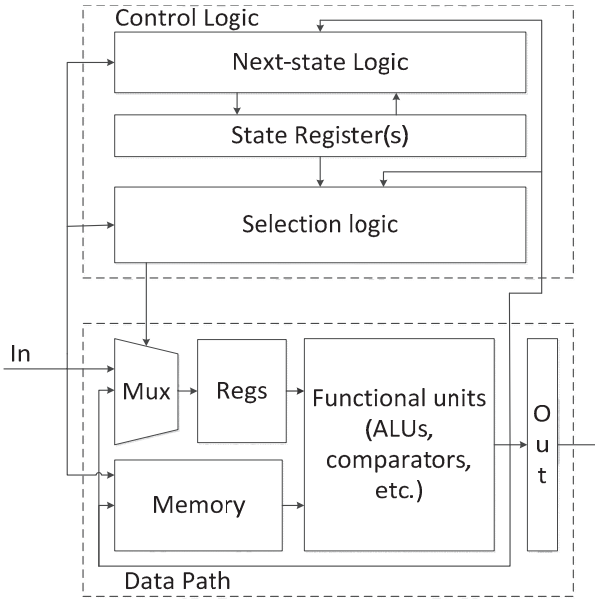

Fig. 1. FSMD view of a circuit.

Finally, in this work we do not consider memory faults, which can be tested by dedicated testing methods, e.g. [15]. Array elements may appear in constraints described in the next section, but we assume that the memory itself is fault free.

III. FSMD-Based Fault Models

The FSMD model contains enough structural information to define a synthesis-agnostic, accurate fault model for the high-level. First, we describe the traditional approach and its correspondence with gate-level SAFs and then augment it with our contribution aimed to detect additional gate-level faults.

A. Traditional high-level fault models.

A number of different fault models were proposed in the literature (e.g. [6-9]). Elaborating these approaches allows us to define the following four fault models each targeting faults at various parts of an FSMD model.

1. Faults at multiplexer select lines. Consider a register $r \in R$. For the next clock cycle it can be assigned to any expression from A_r. Usually, a network of multiplexers is responsible for selecting the correct signal to store at r among the possibilities. Faults in these muxes are modelled as incorrect choices getting into the register. In general, the set of faults is constructed as follows: for each register $r \in R$ and for each two elements $a_c, a_f \in A_r$ a constraint $a_c \neq a_f$ is created where a_c is considered correct and a_f erroneous value.

2. Faults at control branches. Errors at control part may cause the flow to change. This is often modelled as otherwise inactive branch of an "*if*" or "*case*" statement is activated. To propagate such fault to a primary output we need to identify a register that suffers from assigning a value at this incorrect branch and propagate it to a primary output. Sometimes waiting few clock cycles is necessary, until the flow change affects an observable register. To summarize, for each condition C encountered in HLSM, including implicit conditions at else branches, the described faulty behavior is set up and the procedure of finding observable register assignments is executed.

3. State assignment faults. This case is similar to the previous one. For each assignment to the state variable we assume that due to a fault somewhere in the next state logic a wrong value is written. This causes the flow to change and the rest of the procedure is the same as for the model #2.

4. Faults at functional units. At HLSM the mapping of operations to functional modules is not yet available. Thus, in order to achieve the highest coverage it is assumed that each arithmetic or logic operation is computed by a unit dedicated only to this operation. The next limitation is that gate level implementation of this unit is not available, which implies that we must restrict our attention only to its I/O pins. Traditionally (e.g bit coverage model in [9]), erroneous behavior is modelled as stuck-at fault at one of the inputs or outputs. This gives the following fault list: for each assignment of the form $r_1 = f(r_2, \dots, r_n)$, where $r_1, \dots, r_n \in R$, we create fault constraints of a form $f(\dots, r_i, \dots) \neq f(\dots, r_i \vee \overline{m}, \dots)$, $f(\dots, r_i, \dots) \neq f(\dots, r_i \wedge m, \dots)$, $f \neq f \wedge m$ $f \neq f \vee \overline{m}$, where m is a bit mask with the following structure: for each bit position of each function input and output m has this bit unset and all others set and $\overline{m}$ is the complement of m. Unfortunately, while considering only I/O errors is usually enough for logical and shifting operations, arithmetic units tend to have more complex structures where internal faults matter. A gate-level implementation is required to catch them.

The synthesis of arithmetic operations is a well-studied domain and the number of implementation families (e.g. ripple-carry vs. carry-lookahead adder) for a particular operation is limited; this allows us to propose the following solution: for each arithmetic operation create a library of its gate-level implementations and generate tests for each fault of each implementation. While this induces much overhead, fault collapsing methods described in III.C may help to keep it under control. Finally, after the synthesis, when the particular implementation is chosen, the test size can be reduced further.

B. Extending traditional models for higher coverage.

Fault models described above are well established and show good results. However, there are SAFs testable at the gate-level that evade detection at higher level using those models.

Consider the task of testing a network of multiplexers preceding a flip-flop. This network is responsible for choosing a correct signal for the next clock cycle among the number of functional module outputs that potentially can be assigned to this flip-flop. In HLSMs actual multiplexers that comprise the circuitry are not defined yet. In this case models from part *A* of this chapter can help to detect SAFs at data and select inputs, but even in a simplest case of a 2-to-1 multiplexer *there exists a sequence of test vectors that covers all stuck-at-faults at its inputs and output, but misses some of internal faults.*

Consider a 2-to-1 mux with data lines A and B, select line S and output O implementing the function $O = A\overline{S} \vee BS$ (Fig.

2b). There is a fanout point on the select line. When we inject a stuck-at-fault after it, e.g. stuck-at-0 near A, such that A is always selected, the output value will turn into $A \vee BS$. Table I provides 6 test vectors which have the following property: for each SAF at inputs or outputs, the correct output O always differs from the faulty output F, but in presence of our fault of interest the circuit shows the correct behavior. As a result, *in case of a system of multiplexers comprised of individual muxes (Fig. 2a) considering faults at the boundaries is not enough to cover internal faults.*

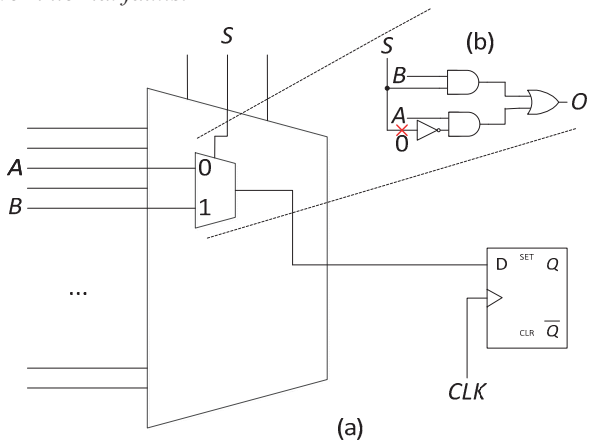

Fig. 2. (a) Multiplexer circuitry with 2-to-1 mux inside. (b) 2-to-1 mux with with stuck-at-0 after fanout.

TABLE I. TESTING 2-TO-1 MUX FAULTS.

Fault	Inputs			Outputs		
	A	B	S	O	F	$A \vee BS$
$A = 0$	1	0	0	1	0	1
$A = 1$	0	0	0	0	1	0
$B = 0$	0	1	1	1	0	1
$B = 1$	0	0	1	0	1	0
$S = 0$	0	1	1	1	0	1
$S = 1$	1	0	0	1	0	1
$O = 0$	1	0	0	1	0	1
$O = 1$	0	0	0	0	1	0

The following observation helps to introduce the model capable to detect such faults. When $A\overline{S} \vee BS$ changes to $A \vee BS$ the erroneous value coming to a flip-flop is $A \vee B$; that is why we propose the term Wired-OR faults for this model.

5. Wired-OR faults for data path. For each register $r \in R$ and for each bit of its range a mask m is created with this bit set and others clear. Then for each $a_c, a_f \in A_r$ and each m the faulty constraint is of a form $a_c \neq a_c \vee (a_f \wedge m)$, which means that one bit of a_c suffers from OR with a faulty bit.

6. Wired-OR faults for control part. This model is intended to increase fault coverage in "Selection logic" box from Fig. 1. In general, its purpose is to capture various conditions, and based on them produce signals that control multiplexers driving the register updates. The plethora of ways such logic can be synthesized does not allow us to be as precise as in model #5 and our analysis is mostly empirical.

Consider the following pattern: **if** C_1 **then** $r := a_1$; **elsif** C_2 **then** $r := a_2$; This can be synthesized in a form $r = C_1 a_1 \vee \overline{C_1} C_2 a_2$. Due to a fault in selection logic both conditions are evaluated to true and again we get $a_1 \vee a_2$ written to r. What makes this situation different from the previous case is, first, the fault occurs before the selection lines are split and considering individual bits is not necessary; second, the assignment $r := a_1$ can be encountered multiple times with different sets of enabling conditions in the design code. In model #5 choosing one of these condition sets is enough while here all of them are investigated.

Hence, for each pair a_c, a_f of assignments to r, and for each set of conditions C that must hold in order to assign a_c to r the fault is defined as a constraint set $C \cup \{a_c \neq a_c \vee a_f\}$.

C. Fault domination and reducing test sizes.

Models #1-6 introduce many new faults, exceeding the number of gate-level counterparts and the ways to reduce them must be considered. Similar to the classical SAF model, fault domination and equivalence relations can be defined.

A fault f is said to *dominate* another fault f' if all tests of f' detect f. Described models capture the erroneous behavior in a form of constraints. Whenever a constraint C_1 implies C_2, we have a domination relation. In this case if test generation for C_1 is succeeded, C_2 can be removed from the fault list; conditions $C_1 \rightarrow C_2$ are easily testable by an SMT-solver. E.g. model #5 faults tend to dominate over model #1 ones.

Another improvement can be achieved by considering multiple faults at a time, e.g. in case of multiple parallel assignments, an ATPG tool targeting a fault at one of them, may check faults at other assignments as a secondary goal.

Finally, static test set compaction methods [16,17] can be applied once the gate-level is synthesized.

IV. EXPERIMENTS

In order to assess the proposed model's ability to cover realistic gate-level faults the following experiment has been set up. First, benchmarks from ITC'99[12] suite were chosen that provide VHDL descriptions of HLSMs as well as gate-level netlists. Strategate [4], as a state-of-the-art freely available academic sequential gate-level test generator, was applied. Tests for the HLSM descriptions of the benchmarks were generated by presented approach. Finally, the fault coverage for all these tests was measured against the fault lists that come with the suite using the fault simulator Hope [11]. Our target for this experiment was to show the capability of proposed fault models to capture the faulty behavior of the circuit at higher levels thereby producing tests of at least the same quality as the stuck-at tests.

Next, in order to demonstrate the advantage of high-level approach, a more complex experiment was conducted. The input for that experiment is a robot controller called RobotBeni which is automatically synthesized from algorithmic description into a HLSM using a high-level synthesis tool Synthagate [13,14]. Then, a gate-level description was produced by Synopsis Design Compiler. This design describes a behavior of a robot whose goal is to reach a target located within a bounded room. It has 3 one-bit inputs that correspond to sensors and 8 one-bit outputs that send signals to wheels whether to move, stop, turn, etc.; its netlist contains 191 flipflops and 1969 gates. The design computes how to move, when to turn etc., in order to reach the goal. The counters are extensively used for syncing the fast chip internal clock with much slower ability of the motor to react on commands.

The selected ITC99 benchmarks can be divided into 3 groups from the test generation perspective. The first one

TABLE II. EVALUATION OF ATPG TOOLS AND FAULT MODELS

design	Strategate			Proposed			speed-up, times	Traditional fault model		loss of FC, %
	time,s	*undetected*	*FC, %*	*time,s*	*undetected*	*FC, %*		*undetected*	*FC, %*	
b01	1.42	0	**100**	34.81	0	**100**	0.04	4	96.46	3.54
b02	0.93	1	**98.39**	9.73	1	**98.39**	0.10	3	95.16	3.23
b03	1628	113	**70.73**	270.15	113	**70.73**	6.03	115	70.21	0.52
b04	27289	136	**91.74**	658.21	136	**91.74**	41.46	155	90.58	1.15
b06	3.77	1	**99.25**	27.33	1	**99.25**	0.14	2	98.51	0.75
b09	247.6	50	**87.59**	517.92	50	**87.59**	0.48	59	85.36	2.23
b11	22356	291	**83.14**	6709	291	**83.14**	3.33	297	82.79	0.35
robot	86400[a]	4082	15.62	5635	2077	**57.07**	15.33	2087	56.86	0.58

[a] Stopped manually after reaching a 24-hours timeout

contains b01, b02 and b06. These are benchmarks where almost all the logic belongs to the control part. The next group contains a sole member b09. A typical operation in this design is reading single bits from register A and copying them to register B, i.e. a serial to parallel conversion. Being single-bit designs, these two groups give no advantage to RTL ATPG in terms of test generation time. Finally, the last category contains larger designs, b03, b04 and b11, which are complex enough to demonstrate the advantages of high-level approach.

Table II presents the comparison between the gate-level ('Strategate'), proposed and traditional fault models. For Strategate and the proposed ATPG the test generation time ('time, s'), the number of undetected faults ('undetected') and the stuck-at fault coverage ('FC, %') is reported. As it can be seen, both ATPGs reach exactly the same coverage for the ITC99 designs. However, the high-level ATPG obtains 3.65 times higher coverage for the RobotBeni example ('robot'). Column ('speed-up, times') shows the speed-up achieved by the proposed approach with respect to STRATEGATE. The three last columns of Table II present the results of a traditional fault model [8]. As it can be seen, the fault coverage achieved is always less than the one by the proposed approach.

V. CONCLUSIONS

The paper proposes a set of dedicated behavioral fault models that allows obtaining fault coverage equal or higher than the one achievable by the best gate-level tools. Additionally, it was shown by the experimental study that previously developed RTL fault models are unable to guarantee high gate-level fault coverage for designs represented at HLSM; it is less than the one by the proposed approach by 1.68% in average. Although not the main focus of this work, the experimental results carried out in this paper confirmed the clear advantage of the high-level ATPG against the gate-level methods in terms of run-time.

ACKNOWLEDGMENT

The work has been supported in part by EU research projects BASTION and IMMORTAL.

REFERENCES

[1] T. Niermann and J. Patel. HITEC: A test generation package for sequential circuits. In *Proc. Eur. DAC*., pages 214–218, Feb. 1991.

[2] X. Chen and M.L. Bushnell, "SEST - A Sequential Circuit Automatic Test Pattern Generator at Rutgers - User's Guide", Rutgers University, Piscataway, NJ, March, 1994.

[3] F. Corno, et al., "GATTO: A Genetic Algorithm for Automatic Test Pattern Generation for Large Synchronous Sequential Circuits", *IEEE Trans.on CAD*, Vol. 15, No 8, 1996, pp. 991-1000.

[4] M. S. Hsiao, E. M. Rudnick, J. H. Patel; "Dynamic state traversal for sequential circuit test generation," ACM TODAES, vol. 5, no. 3, pp. 548-565, July, 2000.

[5] A. Giani, S. Sheng, M. S. Hsiao, V.D. Agrawal, Efficient Spectral Techniques for Sequential ATPG, *Proc. of DATE*, pp 455-464, 2008.

[6] Corno, F. ; Cumani, G. ; Reorda, M.S. ; Squillero, G., Effective techniques for high-level ATPG, Proc.of ATS, pp. 225-230, 2001.

[7] Alizadeh, B. ; Fujita, M., Guided gate-level ATPG for sequential circuits using a high-level test generation approach, *Proceedings of ASP-DAC*, pp. 425-430, 2010.

[8] J. Raik, R. Ubar, T. Viilukas, M. Jenihhin, Mixed hierarchical-functional fault models for targeting sequential cores. Journal of Systems Architecture,Vol. 54, No. 3-4, pp. 465-477, 2008.

[9] Ferrandi, F. ; Ferrara, G. ; Sciuto, D. ; Fin, A. ; Fummi, F. Functional test generation for behaviorally sequential models, Proc. of Design, Automation and Test in Europe, pp. 403-410, 2001.

[10] D.D. Gajski; N.D. Dutt; A.C.-H. Wu; S.Y.-L. Lin, "High-Level Synthesis: Introduction to Chip and System Design", Kluwer Academic Publishers, Dordrecht, 1992.

[11] H. K. Lee; D. S. Ha, "HOPE: an efficient parallel fault simulator for synchronous sequential circuits", *IEEE Trans. on CAD*, vol.15, no.9, pp.1048-1058, Sep 1996.

[12] Corno, F.; Sonza Reorda, M.; Squillero, G., "RT-level ITC'99 benchmarks and first ATPG results," Design & Test of Computers, IEEE, vol.17, no.3, pp.44-53, Jul/Sep 2000.

[13] Synthagate: a high level synthesis tool by Synthezza. http://www.synthezza.com/examples-of-hls/

[14] Baranov, S.; "Logic and System Design of Digital Systems", TUT Press, Tallinn, 2008.

[15] Hamdioui, S.; Al-Ars, Z.; van de Goor, A.J.; Rodgers, M., "Linked faults in random access memories: concept, fault models, test algorithms, and industrial results", *IEEE Trans. on CAD*, vol.23, no.5, pp.737-757, May 2004.

[16] Dimopoulos, M.; Linardis, P., "Efficient static compaction of test sequence sets through the application of set covering techniques," in *Proc. of DATE*, pp.194-199, Feb. 2004.

[17] Pomeranz, I., "Two-Dimensional Static Test Compaction for Functional Test Sequences," in *Computers, IEEE Transactions on* , vol.64, no.10, pp.3009-3015, Oct. 1 2015.

[18] The International Technology Roadmap for Semiconductors, http://www.itrs.net/

[19] Bergamaschi, R.; Kuehlmann, A., "A system for production use of high-level synthesis," in *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, vol.1, no.3, pp.233-243, Sept. 1993.