In Proceedings of the IEEE International Conference on Computer Design (ICCD), pages 510-515, October 1998

An Exact Solution to the Minimum Size Test Pattern Problem

Paulo F. Flores, Horácio C. Neto and João P. Marques Silva
Cadence European Laboratories
IST/INESC
1000 Lisboa, Portugal
{pff,hcn,jpms}@inesc.pt

Abstract

This paper addresses the problem of test pattern generation for single stuck-at faults in combinational circuits, under the additional constraint that the number of specified primary input assignments is minimized. This problem has different applications in testing, including the identification of don't care conditions to be used in the synthesis of Built-In Self-Test (BIST) logic. The proposed solution is based on an integer linear programming (ILP) formulation which builds on an existing Propositional Satisfiability (SAT) model for test pattern generation. The resulting ILP formulation is linear on the size of the original SAT model for test generation, which is linear on the size of the circuit. Nevertheless, the resulting ILP instances represent complex optimization problems, that require dedicated ILP algorithms. Preliminary results on benchmark circuits validate the practical applicability of the test pattern minimization model and associated ILP algorithm.

1. Introduction

Automatic test pattern generation (ATPG) for stuck-at faults in combinational circuits is now a mature field, with an impressive number of highly effective models and algorithms [5-7, 12-14]. (A comprehensive list of references can be found in [4].) Furthermore, besides being effective at detecting the target faults, recent ATPG tools have aimed the heuristic minimization (i.e. compaction) of the total number of test patterns required for detecting all faults in a circuit [3, 11, 12]. In general, the degree of test pattern compaction is expected to be related to the number of unspecified input assignments in each test pattern. In addition, for applications where testing time and fault coverage requirements can only be obtained with dedicated Finite-State Machine (FSM) controllers, the computation of test patterns with a large number of unspecified input assignments may allow for significantly smaller synthesized FSMs. Indeed, if the test set is used as input to a logic synthesis tool with the purpose of synthesizing BIST logic, then by maximizing the number of unspecified input assignments, i.e. by maximizing the don't care set of each test pattern, the logic synthesis tool is in general able to yield smaller synthesized logic. Thus the maximization of the don't care set of each test pattern, or conversely, the computation of test patterns of minimum-size, can have significant practical consequences.

Nevertheless, there exists no model or algorithm in the literature for computing test patterns for which the number of unspecified primary input assignments is maximized. Accordingly, the main objective of this paper is to propose a first attempt at solving this problem. We start by formalizing the notion of test pattern minimization. We then develop a new model for test pattern generation, based on propositional satisfiability (SAT), in the presence of unspecified input assignments. Next, we derive an integer linear programming (ILP) model for maximizing the number of unspecified primary input assignments. Afterwards, we show that the model is indeed correct and analyze some of its limitations. Finally, we provide preliminary results that justify using the proposed model in medium-size combinational circuits and describe an ATPG methodology, which can incorporate the proposed model and supporting algorithm, and which can also be applied to large-size combinational circuits. Besides its practical applicability, to our best knowledge this is the first formal non-heuristic model towards computing minimum size test patterns.

The paper is organized as follows. We start in Section 2 with several definitions regarding combinational circuits, Conjunctive Normal Form (CNF) representations of circuits and CNF representations of fault detection problems, which are used throughout the paper. Afterwards, in Section 3, the CNF models described in Section 2 are generalized for correctly handling unspecified variable assignments. The next step is to introduce the ILP optimization model for minimizing test patterns and prove its correctness. Section 5 includes preliminary experimental results on several practical applications of the model. We conclude in Section 6 with a brief overview of future research work in the area of test pattern minimization.

2. Definitions

We start by introducing unified representations for circuits, fault detection problems, and associated optimization problems. These representations are used throughout the paper and are key for developing the proposed ILP formulations. A combinational circuit C is represented as a directed acyclic graph $C = (V_C, E_C)$, where the elements of V_C, i.e. the circuit nodes, are either primary inputs or gate outputs, with $|V_C| = N$. The set of edges $E_C \subseteq V_C \times V_C$ identifies gate input-output connections. We shall assume gates with bounded fanin, and so $|E_C| = O(|N|)$. For every circuit node x in V_C, the following definitions apply (from [13]):

- $I(x)$ denotes the ***fanin*** nodes of node x, i.e. nodes y in V_C such that $(y, x) \in E_C$.
- $O^*(x)$ denotes the ***transitive fanout*** of node x, i.e. the set of all nodes y such that there is a path connecting x to y.

- $K_O(x)$ denotes ***immediate fanout cone of influence*** of x, being defined as follows:

$$K_O(x) = \{y | y \in O^*(x) \vee y \in I(w) \wedge w \in O^*(x)\} . \quad (1)$$

The set of primary inputs is referred to as *PI*, and the set of primary outputs as *PO*. Simple gates are assumed: AND, NAND, OR, NOR, NOT and BUFF.

For Automatic Test Pattern Generation (ATPG), the following definitions apply. The single stuck-at line (SSF) fault model is assumed [1]. We say that a stuck-at fault is ***detectable*** if and only if there exists an assignment of logic values to the circuit primary inputs such that the effect of the fault can be observed at one of the circuit primary outputs.

The application of Conjunctive Normal Form (CNF) representations of circuits and fault detection problems in ATPG has been extensively studied [6, 13, 14]. In this section we provide very simple and non-optimized CNF representations of circuits and fault detection problems, which will be assumed in the remainder of the paper. These representations form the basis for the ILP models introduced in the remainder of the paper.

The CNF formula of a circuit is the conjunction of the CNF formulas for each gate output, where the CNF formula of each gate denotes the valid input-output assignments to the gate. (Derivation of the CNF formulas for simple gates can be found for example in [6, 13].) If we view a CNF formula as a set of clauses, the CNF formula φ for the circuit is defined by the set union[1] of the CNF formulas for each gate:

$$\varphi = \bigcup_{x \in V_C} \varphi_x \quad (2)$$

In the context of test pattern generation, and for capturing the fault detection problem, each node x is characterized by three propositional variables:

- x^G denotes the logic value assumed by the node in the ***good*** circuit.
- x^F denotes the logic value assumed by the node in the ***faulty*** circuit.
- x^S denotes whether x^G and x^F assume different logic value [6]. We shall refer to this variable as the ***sensitization status*** of node x. (Other semantic definitions of the sensitization status have been proposed [14]. Nevertheless, the above definition is used since it simplifies the ILP formulations derived in subsequent sections. We should note, however, that the other semantic definitions could also be used.)

Given the definition of variable x^S, the condition $\left[\left(x^G \neq x^F\right) \leftrightarrow x^S\right]$ must hold, which can be simplified to:

$$\begin{aligned}\varphi_x^S = &\left(x^G + \neg x^F + x^S\right) \cdot \left(\neg x^G + x^F + x^S\right) \cdot \\ &\left(\neg x^S + x^G + x^F\right) \cdot \left(\neg x^S + \neg x^G + \neg x^F\right)\end{aligned} \quad (3)$$

that basically states that the logic values of x^G and x^F differ if and only if x^S assumes logic value 1.

1. Set union in this context is to be understood as a product of clauses.

Let φ_x denote the CNF formula associated with gate output x. The notation φ_x^G denotes the CNF formula for x in the good circuit, i.e. using y^G variables, whereas φ_x^F denotes the CNF formula for x in the faulty circuit, i.e. using y^F variables. For a ***stem*** fault z-a-v [1], the CNF representation of the associated fault detection problem contains the following components:

- CNF formula φ^G denoting the good circuit.
- CNF formula φ^F denoting the faulty circuit. This formula only needs to contain the CNF formulas for the nodes that are relevant for detecting the given fault, i.e. nodes in the transitive fanout of node z.
- CNF formulas φ^S for defining the sensitization status of every node in the transitive fanout of the fault site, i.e. node z. Hence, for each of these nodes add φ_x^S, which states that $x^S = 1$ if and only if $x^G \neq x^F$.
- Clauses φ^B that prevent each node x from being sensitized, by having $x^S = 0$, whenever x is not in the transitive fanout of z but at least one fanout node of x is in the transitive fanout of z, i.e. x is in $K_O(z) - O^*(z)$.
- Clauses requiring $x^G = x^F$ on each node x such that x is not in the transitive fanout of z but at least one fanout node of x is in the transitive fanout of z, i.e. x is in $K_O(z) - O^*(z)$. (Observe that this condition and the previous one permit restricting the number of x^F and x^S variables that must actually be used.)
- Clauses φ^A capturing conditions for ***activating*** the fault, i.e. by requiring $z^G \neq z^F$ and by forcing a suitable logic value on z^G.
- Clause φ^R requiring that at least one sensitization variable of a primary output in the transitive fanout of the fault site assumes value 1.

(A more detailed derivation of the union of the previous sets of clauses, φ^D, for detecting a fault z s-a-v can be found in [13].) φ^D will henceforth be referred to as the ***fault detection formula***. Finally, we observe that a similar model can be constructed for ***fanout-branch*** faults [6, 13].

The proposed CNF formulations can be simplified and improved (see for example [6, 13, 14] for further details). Nevertheless, for the purposes of this paper the proposed formulation suffices and shall be assumed.

3. Test Generation With Don't Cares

The SAT-based test generation model described in the previous section requires all clauses to be satisfied, hence most if not all variables must be assigned a logic value. However, we want to develop a test generation model that properly handles unspecified variable assignments, since our goal is to compute minimum size test patterns. As a result, in this section we develop models for circuit satisfiability and test generation using CNF formulas that can be satisfied in the presence of unspecified variable assignments.

3.1. Modeling Unspecified Variable Assignments

Given a circuit and its associated CNF formula or a fault f and its associated fault detection formula, the exist-

x	0	1	X
$\left(x^0, x^1\right)$	(1, 0)	(0, 1)	(0, 0)

Figure 1: Modeling unspecified assignments

ence of unspecified assignments implies that each of the original circuit variables can now be assigned a value in the set $\{0, 1, X\}$. In this situation an assignment $x = X$ indicates that x is ***unspecified***, or that the value assumed by x is an unspecified assignment. In contrast $x \in \{0, 1\}$ indicates that x is ***specified***, or that the value assumed by x is a specified assignment. In this situation, an assignment A is allowed to leave variables unspecified. Furthermore, the value of a CNF formula φ for an assignment A can also be X, $\varphi|_A \in \{0, 1, X\}$.

With the purpose of deciding CNF formula satisfiability, in the presence of unspecified variables, a new set of variables is created. This basically consists of duplicating the number of Boolean variables, which is a common solution for capturing unspecified assignments [10]. (Observe that since only 3^M assignments need to be considered for M variables, the actually required number of Boolean variables is $\lceil \log\left(3^M\right) \rceil$, since there are only three possible assignments to each of the original variables. Nevertheless, considering instead $2M$ variables greatly simplifies the proposed model.) As a result, we propose to represent each Boolean variable x with two new variables x^0 and x^1 having the interpretation indicated in Figure 1. For this interpretation, $x = X$ indicates that x is unspecified. The simultaneous assignment of variables x^0 and x^1 to 1 is not allowed, requiring the inclusion of the following constraints in the resulting CNF formula,

$$\varphi_{inv,\,x} = \left(\neg x^1 + \neg x^0\right) \tag{4}$$

for each node $x \in V_C$, where V_C represents the set of nodes in the circuit. In addition, for each basic gate type we need to define the corresponding CNF formula. However, using the ideas above, each gate input and output must now be replaced by two variables. Let us consider for example an AND gate, which will now be denoted by the generalized form $\left(x^0, x^1\right) = \text{UAND}\left(w_1^0, w_1^1, \ldots, w_j^0, w_j^1\right)$, and which allows unspecified assignments to the gate inputs and output. Since the simultaneous assignment of any pair of variables $\left(x^0, x^1\right)$ to 1 is prevented by (4), then we just need to relate the remaining assignments. The output variable x^1 can only assume value 1 whenever all input variables w_j^1 also assume value 1. Hence, we can say that $x^1 = \text{AND}\left(w_1^1, \ldots, w_j^1\right)$. In addition, the output variable x^0 assumes value 1 provided at least one input variable w_j^0 assumes value 1. Hence, we can say that $x^0 = \text{OR}\left(w_1^0, \ldots, w_j^0\right)$. As a result we obtain from [6, 13],

$$\begin{aligned} \varphi_{u,\,x^0} &= \left[\prod_{i=1}^{j}\left(\neg w_i^0 + x^0\right)\right] \cdot \left(\sum_{i=1}^{j} w_i^0 + \neg x^0\right) \\ \varphi_{u,\,x^1} &= \left[\prod_{i=1}^{j}\left(w_i^1 + \neg x^1\right)\right] \cdot \left(\sum_{i=1}^{j} \neg w_i^1 + x^1\right) \end{aligned} \tag{5}$$

x^G	X	0	0	1	1	1	0	X	X
x^F	X	0	1	0	1	X	X	0	1
x^S	0	0	1	1	0	0	0	—	—

Figure 2: Truth table for the sensitization status

Furthermore, the CNF formula for an AND gate with output x now becomes,

$$\varphi_{u,\,x} = \varphi_{u,\,x^1} \cup \varphi_{u,\,x^0} \cup \varphi_{inv,\,x} \tag{6}$$

which properly models unspecified assignments to the inputs and output of an AND gate. Similar relations can be derived for the other simple gates. Consequently, the CNF formulas for simple gates given in [13] can be generalized by following the same approach used for deriving (5) as shown in [4]. As a result, and as was done in Section 2, we can now create the CNF formula for the circuit, one in which unspecified variable assignments are allowed.

3.2. Test Generation with Don't Cares

We can now generalize the test pattern generation model of Section 3 so that unspecified variable assignments are allowed. Each circuit node x is still characterized by three variables:

- x^G denoting the value in the good circuit. This variable can be unspecified, and so we use two new variables to characterize its value, $x^{G,0}$ and $x^{G,1}$, with the semantic definition given earlier.
- x^F denoting the value in the faulty circuit. This variable can also be unspecified, and so we use two new variables to characterize its value, $x^{F,0}$ and $x^{F,1}$, with the semantic definition given earlier.
- x^S denoting the sensitization status of each node. As we will justify below, the sensitization status of each node needs not be unspecified, and so its value is always either 0 or 1.

Modeling unspecified assignments in test generation requires a detailed characterization of the propagation of the fault effect. Hence, the sensitization status x^S of a node can only assume value 1 when both the values of node in the good and faulty circuits are *specified* and assume different logic values. Moreover this requirement also causes the value of a node in the faulty circuit to be specified *only* when the value of that node in the good circuit is also specified. These constraints indicate that propagation of the fault effect to a node can only be guaranteed when the values in the good and faulty circuit are specified for that node.

Consequently, the relationship between the value of x^S and the possible values of x^G and x^F is shown in Figure 2. Entries with a '—' denote invalid value assignments, for which the CNF formula for x^S must assume value 0. Similarly to the model for completely specified assignments, x^S assumes value 1 if and only if x^G and x^F assume opposing logic values, provided that both x^G and x^F are

specified. The simplification of the truth table in Figure 2 yields the following CNF formula for the sensitization status of node x, x^S:

$$\begin{aligned}\varphi^S_{u,x} = &\left(x^{G,1} + x^{G,0} + \neg x^S\right)\cdot\left(x^{F,1} + x^{F,0} + \neg x^S\right)\cdot\\ &\left(x^{F,1} + x^{G,1} + \neg x^S\right)\cdot\left(\neg x^{G,1} + \neg x^{F,1} + \neg x^S\right)\cdot\\ &\left(x^{G,1} + \neg x^{F,1} + x^S\right)\cdot\left(x^{G,0} + \neg x^{F,0} + x^S\right)\end{aligned} \quad (7)$$

The next step is to describe the modifications to the CNF formula used for computing the faulty values, which for completely specified assignments are equivalent to the CNF formula for the good value. For incompletely specified assignments the same holds true but, as justified above, we introduce the additional constraint that an unspecified good value implies and unspecified faulty value,

$$\left(x^G = X\right) \Rightarrow \left(x^F = X\right) \quad (8)$$

Let us assume that the CNF formula for the faulty value of a node x with completely specified assignments is given by,

$$\varphi^F_x = \prod_{i=1}^{j} \omega_i \quad (9)$$

As a result of (8), the CNF formula for the faulty circuit, in the presence of incompletely specified assignments, is defined by,

$$\begin{aligned}\varphi^F_{u,x} &= \left(\neg x^{F,0} + x^{G,0} + x^{G,1}\right)\cdot\left(\neg x^{F,1} + x^{G,0} + x^{G,1}\right)\cdot\\ &\quad\cdot\prod_{i=1}^{j}\left(\omega_i + \neg x^{G,0}\cdot\neg x^{G,1}\right)\cdot\\ &= \left(\neg x^{F,0} + x^{G,0} + x^{G,1}\right)\cdot\left(\neg x^{F,1} + x^{G,0} + x^{G,1}\right)\cdot\\ &\quad\cdot\prod_{i=1}^{j}\left[\left(\omega_i + x^{G,1}\right)\cdot\left(\omega_i + x^{G,0}\right)\right]\cdot\end{aligned} \quad (10)$$

Hence, the faulty value of a node x is computed by its original formula provided the good value is specified (i.e. $x^{G,0} + x^{G,1} = 1$). In contrast, if the good value is unspecified (i.e. $x^{G,0} + x^{G,1} = 0$), then the faulty value is *forced* to also be unspecified.

The formulas for $\varphi^S_{u,x}$ and for $\varphi^F_{u,x}$ are defined so that an unspecified good value immediately implies an unspecified faulty value and $x^S = 0$. Thus propagation of the error signal is only permitted in the presence of properly specified values for the good circuit variables.

Furthermore, we note that the remaining CNF formulas, i.e. propagation blocking conditions φ^B and fault detection requirements φ^R, remain unchanged, whereas the fault activation conditions φ^A must be updated to the new set of variables. As a result the complete CNF formula for a given stem fault z s-a-v is summarized in Table 1. Similarly, we can derive the CNF formula for a fanout-branch fault. Furthermore, we refer to φ^D_u as the fault-detection formula in the presence of unspecified variable assignments. Consequently, and when referring to primary input assignments, or test patterns, we must now assume that some primary inputs may be unspecified.

Sub-formula/ Condition	Clause Set
Good Circuit	$\varphi^G_u = \bigcup_{x \in V_C} \varphi^G_{u,x}$
Faulty Circuit	$\varphi^F_u = \bigcup_{x \in O^*(z)} \varphi^F_{u,x}$
Node Sensitization	$\varphi^S_u = \bigcup_{x \in O^*(z)} \varphi^S_{u,x}$
Block Propagation	$\varphi^B_u = \left(\neg x^S\right) \quad x \in K_O(z) - O^*(z)$
Fault Activation	$\varphi^A_u = \left(z^S\right)\cdot\left(\neg z^{G,1}\right)\cdot\left(z^{G,0}\right)\cdot\left(z^{F,1}\right)\cdot\left(\neg z^{F,0}\right)$
Require Detection	$\varphi^R_u = \left(\sum_{x \in PO \wedge x \in O^*(z)} x^S\right)$
Detection Formula	$\varphi^D_u = \varphi^G_u \cup \varphi^F_u \cup \varphi^S_u \cup \varphi^B_u \cup \varphi^A_u \cup \varphi^R_u$

Table 1: Fault detection formula for fault z s-a-v ($v = 1$)

4. Computing Minimum Size Test Patterns

In this section we develop the optimization model for computing minimum-size test patterns. This optimization model is based on test pattern generation in the presence of incompletely specified primary input assignments. Moreover, stem faults are assumed throughout, even though the same approach is readily applied to fanout-branch faults.

The main objective of test pattern minimization is to identify the minimum number of primary input assignments which detect the fault. Hence, our goal is to minimize the number of specified primary input assignments such that the given fault is still detected. As a result we obtain the following optimization model,

$$\begin{aligned}&\text{minimize} && \sum_{x \in PI}\left(x^0 + x^1\right)\\ &\text{subject to} && \varphi^D_u\end{aligned} \quad (11)$$

which basically requires that the total number of assigned input variables be minimized under the constraint that the fault be detected. (Observe that we have $0 \le x^0 + x^1 \le 1$ given (4), which implies an upper bound on the value of the cost function of $|PI|$.) Given the mapping between CNF clauses and linear inequalities [10] we immediately conclude that (11) corresponds to an integer linear program, and so different integer linear optimization packages can be used for solving the test pattern minimization problem. Nevertheless, the constraints of (11) are tightly related with propositional satisfiability. Consequently, and as shown in [9], SAT-based ILP solvers are preferable for solving ILPs for which the constraints correspond to CNF formulas. For the experimental results given in Section 5,

the SAT-based ILP solver of [9] was used.

The validity of the proposed optimization model is formally established in [4]. We should note, however, that in general there may exist faults for which it is possible to identify test patterns with a smaller number of specified assignments, but which do not uniquely identify a set of sensitizable paths. A simple example is a multiplexer [4]. Consequently, a test generation model based on the *D*-calculus [1] or any of its derivations is by itself unable to identify *all* test patterns which do not uniquely identify a set of sensitizable paths, since for some cases propagation does not actually take place and only the propagation conditions are implicitly validated. As a result, our proposed model yields the minimum-size test patterns which guarantee, given the specified assignments, propagation of the fault effect to a primary output by defining one or more sensitizable paths.

5. Experimental Results

The model described in the previous section has been integrated in a test pattern generation framework for the computation of minimum size test patterns referred to as *Minimum Test Pattern generator* (MTP), which uses the SAT-based ILP algorithm of *bsolo* [9] and the fault simulator provided with ATALANTA [7]. The results included below were obtained with the IWLS'89 benchmark suite [8] and with the ISCAS'85 benchmark suite [2]. In all cases MTP was run with a bound on the amount of allowed search (i.e. the total number of conflicts [9]). This permits MTP to identify acceptable solutions, which in some cases may not be necessarily optimal. Moreover, in order to speed up convergence to the optimal solutions, MTP uses the solution computed by ATALANTA (or by any other ATPG tool) as the startup assignment. These assignments provide an initial upper bound on the value of the optimal solution. If ATALANTA aborts the fault, then TG-GRASP [13] is used for computing a startup test pattern.

Table 2 contains the results for the IWLS'89 benchmarks for both ATALANTA [7] and MTP. ATALANTA is an ATPG tool that can generate test patterns with don't cares. For each benchmark *all* faults were targeted in order to allow for a a meaningful comparison between the two algorithms. Columns #PI, #G, #F, #R and #A denote, respectively, the number of primary inputs, gates, faults, redundant faults and aborted faults. %X denotes the percentage of don't care bits in all test patterns; Δ denotes the variation in percentage from ATALANTA to MTP; %Opt denotes the percentage of faults for which MTP was able to find the actual minimum-size test pattern. Finally, time/ fault denotes the average time spent solving the ILP for each fault.

From these results several conclusions can be drawn. First, MTP allows validating the heuristics used in ATALANTA for computing test patterns with don't cares. Indeed for several benchmarks, ATALANTA already identifies the minimum-size test patterns for all faults. Nevertheless,

Circuit	#*F*	ATALANTA			MTP					
		#R	#A	%X	#R	#A	%X	Δ	%opt	time/ fault
9symml	752	2	0	1.4	2	0	8.9	7.5	100	2.04
cht	820	0	0	93.6	0	0	94.4	0/8	100	0.64
cm138a	124	0	0	16.7	0	0	16.7	0.0	100	0.02
cm150a	232	0	0	68.4	0	0	71.0	2.6	100	1.55
cm163a	220	0	0	70.7	0	0	72.8	2.1	100	0.28
cmb	248	0	0	29.6	0	0	30.0	0.4	100	0.07
comp	480	1	0	24.0	1	0	39.6	15.6	2	10.64
comp16	960	0	0	30.7	0	0	32.9	2.2	4	13.66
cordic	342	0	0	30.7	0	0	40.2	9.5	37	6.28
cu	262	7	0	53.0	7	0	57.1	4.1	100	0.14
majority	54	0	0	8.5	0	0	8.5	0.0	100	0.01
misex1	224	0	0	49.8	0	0	54.4	4.6	100	0.17
misex2	422	0	0	73.5	0	0	75.8	2.3	100	0.20
misex3	2590	7	0	24.4	7	0	37.7	13.3	76	25.29
mux	202	0	0	67.3	0	0	75.8	8.5	100	0.94
pcle	328	0	0	73.3	0	0	74.9	1.6	99	0.45
pcler8	400	0	0	78.1	0	0	79.2	1.1	98	1.97
term1	708	6	0	72.2	6	0	74.4	2.2	86	4.35
too_large	1132	15	0	54.9	15	0	62.2	7.3	20	18.27
unreg	448	0	0	90.6	0	0	91.7	1.1	86	0.93

Table 2: Experimental results for the IWLS'89 circuits

for other benchmarks, the test patterns computed by ATALANTA can be far from the minimum-size test patterns. For these cases the percentage of don't cares computed with MTP can be as much as 15% above the values computed by ATALANTA. Finally, we observe that for medium-size circuits MTP is able to compute the actual minimum-size test patterns for all faults in the circuit in a reasonable amount of time per fault. For larger circuits, MTP finds solutions that are better than those computed by ATALANTA, but which are not guaranteed to be optimal.

Table 3 contains the results for the ISCAS'85 circuits[2]. For these benchmarks a smaller search effort (i.e. 100 conflicts) was allowed. This leads to smaller run times and, consequently, less optimal results. Once more we can conclude that MTP is able to improve over the ATALANTA results, but in this case the improvements are in general smaller, since it becomes harder for the ILP solver [9] to find optimal solutions. (As can be concluded the percentage of optimal solutions found ranges from 0 to 20 percent.) For some of these circuits we run MTP with a larger number of allowed conflicts (i.e. 1000 conflicts). The obtained results are shown in Table 4. As can be observed, a larger percentage of unspecified input assignments is obtained at the cost of a larger search effort per fault. Accordingly, the time per fault also increases.

2. Observe that ATALANTA aborts several faults for c432, c2670, c6288 and c7552. For those cases, MTP uses TG-GRASP [13] as the startup ATPG tool, and consequently does not abort any fault.

Circuit	#F	ATALANTA			MTP					
		#R	#A	%X	#R	#A	%X	Δ	%opt	time/ fault
c432	524	3	1	56.2	4	0	60.8	4.6	0	3.21
c499	758	8	0	17.1	8	0	18.7	1.6	0	4.35
c880	942	0	0	82.2	0	0	83.8	1.6	12	2.54
c1355	1574	8	0	13.3	8	0	13.7	0.4	0	9.12
c1908	1878	8	0	44.7	8	0	48.4	3.7	0	9.61
c2670	2746	97	20	92.0	117	0	92.4	0.4	23	10.99
c3540	3425	134	0	74.6	134	0	77.3	2.7	15	16.81
c5315	5350	59	0	92.6	59	0	92.9	0.3	14	9.34
c6288	7744	34	387	22.2	34	0	25.1	2.9	1	36.65
c7552	7550	77	181	86.9	131	0	86.9	0/0	4	17.46

Table 3: Results for the ISCAS'85 benchmarks

Circuit	#F	ATALANTA			MTP					
		#R	#A	%X	#R	#A	%X	Δ	%opt	time/ fault
c432	524	3	1	56.2	4	0	64.1	7.9	2	27.04
c499	758	8	0	17.1	8	0	19.5	2.4	0	33.71
c880	942	0	0	82.2	0	0	85.6	3.4	40	22.34
c1355	1574	8	0	13.3	8	0	15.2	1.9	0	64.86
c1908	1878	8	0	44.7	8	0	60.0	15.3	1	73.44
c2670	2746	97	20	92.0	117	0	93.0	1.0	25	83.46

Table 4: Results for the ISCAS'85 circuits

From the previous experimental results for the IWLS'89 and ISCAS'85 benchmarks we can draw the following conclusions:

- For some circuits the heuristics used by ATALANTA, as well as by other structural ATPG algorithms, are extremely effective and MTP can be used to formally prove this result.
- Whenever the main goal is maximizing the number of don't care bits, then MTP can be run on top of ATALANTA (or any other ATPG algorithm), thus in general allowing for an increased number of unspecified bit assignments. The improvements obtained by MTP are related to the amount of allowed search effort, and MTP is always guaranteed to produce results that are no worse than the startup tool (in our case ATALANTA or TG-GRASP).

6. Conclusions

In this paper we introduce a SAT-based integer linear programming model for computing minimum-size test patterns. The applicability of the model has been illustrated by computing minimum size test patterns for several benchmark circuits. The next step of this work is to study the application of minimum-size test patterns to the synthesis of BIST logic, with the objective of evaluating the reduction in size of the synthesized logic obtained from using MTP.

Additional research work involves further constraining the ILP formulation so that larger problem instances can be solved optimally. Furthermore, the tradeoffs between minimum-size test pattern computation, fault simulation and fault compaction need to be studied. Finally, a long-term objective of this work is the integration of the proposed model in a complete testing environment, thus enabling the use of minimum-size test patterns for different purposes, such as the validation of test pattern minimization heuristics or the synthesis of reduced-size FSMs for BIST in specific target applications.

References

[1] M. Abramovici, M. A. Breuer and A. D. Friedman, *Digital Systems Testing and Testable Design*, Computer Science Press, 1990.

[2] F. Brglez and H. Fujiwara, "A Neutral List of 10 Combinational Benchmark Circuits and a Target Translator in FORTRAN," in *Proc. of ISCAS*, 1985.

[3] K. Chakrabarty, B. T. Murray, J. Liu and M. Zhu, "Test Width Reduction for Built-In Self Testing", in *Proc. of ITC*, November 1997.

[4] P. Flores, H. Neto and J. Marques Silva, "An Exact Solution to the Minimum-Size Test Pattern Problem," in *IEEE/ACM IWLS*, June 1998.

[5] P. Goel, "An Implicit Enumeration Algorithm to Generate Tests for Combinational Logic Circuits," *IEEE Transactions on Computers*, vol. 30, no. 3, pp. 215-222, March 1981.

[6] T. Larrabee, "Test Pattern Generation Using Boolean Satisfiability," *IEEE Transactions on Computer-Aided Design*, vol. 11, no. 1, pp. 4-15, January 1992.

[7] H. K. Lee and D. S. Ha, "On the Generation of Test Patterns for Combinational Circuits," Tech. Rep. No. 12_93, Dept. of Electrical Engineering, Virginia Polytechnic Institute and State University, 1993.

[8] IWLS'89 Benchmark Suite, available from http://www.cbl.ncsu.edu/pub/Benchmark_dirs/LGSynth89/.

[9] V. Manquinho, P. Flores, J. P. M. Silva and A. Oliveira, "Prime Implicant Computation Using Satisfiability Algorithms," in *Proc. of ICTAI*, November 1997.

[10] C. Pizzuti, "Computing Prime Implicants by Integer Programming," in *Proc. of ICTAI*, November 1996.

[11] I. Pomeranz, L.N. Reddy, S.M. Reddy, "COMPACTEST: A Method to Generate Compact Test Sets for Combinational Circuits," in *IEEE Transactions on Computer-Aided Design*, vol. 12, no. 7, pp. 1040-1049, July 1993.

[12] M. H. Schulz, E. Trischler and T. M. Sarfert, "SOCRATES: A Highly Efficient Automatic Test Pattern Generation System," *IEEE Transactions on Computer-Aided Design*, vol. 7, no. 1, pp. 126-137, January 1988.

[13] J. P. M. Silva and K. A. Sakallah, "Robust Search Algorithms for Test Pattern Generation," in *Proc. of FTC*, June 1997.

[14] P. R. Stephan, R. K. Brayton and A. L. Sangiovanni-Vincentelli, "Combinational Test Generation Using Satisfiability," *IEEE Transactions on Computer-Aided Design*, vol. 15, no. 9, pp. 1167-1176, September 1996.