TEST PATTERN GENERATION FOR WIDTH COMPRESSION IN BIST

Paulo Flores , *Horácio Neto* — Technical University of Lisbon, IST/INESC, {pff,hcn}@inesc.pt

Krishnendu Chakrabarty — Duke University, Electrical and Computer Eng., krish@ee.duke.edu

João Marques-Silva — Cadence European Laboratories, IST/INESC, jpms@inesc.pt

ABSTRACT

The main objectives of Built-In Self Test (BIST) are the design of test pattern generator circuits which achieve the highest fault coverage, require the shortest sequence of test vectors and utilize the minimum circuit area. This paper targets the problem of generating test patterns for stuck-at faults that induce compatibility relations between the primary inputs of the circuit under test. These compatibility relations can be used for designing counter-based test generator circuits with a reduced number of bits, thus requiring smaller testing time and smaller area. The proposed solution is based on an integer linear programming (ILP) formulation that builds on existing Propositional Satisfiability (SAT) models for test pattern generation. An ATPG tool for minimum test pattern generation for width compression (MTP-C) is described, which illustrates the practical applicability of our approach for a wide range of benchmark circuits.

1. INTRODUCTION

Built-In Self-Test (BIST) denotes the ability of a circuit (or system) to test itself. This paradigm for testing ICs is gaining acceptance in the VLSI industry because it can potentially eliminate the need for external test equipment and introduces the capability for testing devices after the circuit is integrated in a system, in the field (on-line testing) [1]. Test vectors are generated on the chip by a test pattern generator circuit (TPG) and the circuit responses are examined by an output response analyzer (ORA) that determines about the correct operation of the IC [1, 2]. The increase of electronics in safety critical applications also demands the use of on-line testing. Such systems in general require testing to have a high fault coverage and be as fast as possible.

One of the key challenges in BIST is the design of the TPG. An optimal TPG will generate the minimum number of vectors (to reduce testing time) that guarantee the highest fault coverage while introducing the minimum area overhead and performance penalty in the circuit. All these design goals are difficult to meet simultaneously, and several architectures for TPG have been proposed [1]. Two opposite architectures with respect to area overhead and testing time are the ROM based architectures and the counter-based architectures. The former architectures use a ROM to store the vectors generated by an Automatic Test Pattern Generator (ATPG). Thus, high fault coverages and short testing times can be achieved. Conversely, the area overhead introduced by this method (ROM, counter, address decoder, etc.) is in general prohibitive for practical applications. In counter-based architectures the test patterns

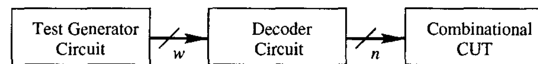

Figure 1: Generic test pattern generator model [4, 2]

are generated by a counter, which introduces a small area penalty. The main disadvantage of this method is that long test sequences may be required to achieve an acceptable fault coverage, which result in longer testing time. The most used BIST architectures are based on Linear Feed Back Registers (LFSR), that are used to generate pseudo-random test sequences. For these architectures, good fault coverages can be achieved in most cases, and the testing time can become sufficiently reduced [3]. Several techniques have been proposed to reduce even more the test time and/or increase the fault coverage based on LFSR architectures [2, 4, 5, 6, 7, 8, 9, 10].

In this paper we describe a model for ATPG targeting counter-based TPG architectures. Constraints are imposed during test pattern generation which target the reduction of width of the counter used in the test generation circuit. Therefore a high fault coverage is guaranteed (100% of non-redundant faults) in a short test time. 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 [11, 12]. The paper is organized as follows. We start in section 2 by identifying our target TPG model and the width compression technique that our work is based upon. Afterwards, in section 3 we address the problem of computing test patterns with unspecified input assignments. Don't cares are of key relevance for identifying compatibility relations between primary inputs of the circuit under test. In section 4 we introduce the ILP test generation model that targets width compression. Section 5 includes preliminary experimental results on several practical applications of the model.

2. BIST CIRCUIT GENERATOR

A large number of techniques exist for designing test generator circuits for BIST [1, 4, 13]. A general model for a BIST scheme is shown in Figure 1. Common solutions for the generation of a set of pre-computed test patterns use a finite state machine (FSM) or a read only memory (ROM) associated with a counter. However, the area overhead introduced by these methods is in general considered prohibitive for practical use.

The most hardware-efficient sequence generator is an LFSR or other counter-like circuit. In these circuits, the number of flip-flops is, in general, equal to the number of inputs of the circuit under test (CUT), thus the decoder circuit is inexistent ($w = n$). These circuits are used in basic pseudo-random or pseudo-exhaustive testing. The fault coverage is determined by fault simulation once the

*This work was partial supported by following PRAXIS XXI projects: Euro-Lasic (2/2.1/TIT/1643/95) and GRASP (2/2.1/TIT/1597/95).

0-7803-5471-0/99/$10.00©1999 IEEE

In Proceedings of the IEEE International Symposium on Circuits and Systems (ISCAS), volume I, pages 114-118, May 1999

test generator circuit is defined. In general, long test sequences are necessary to achieve high fault coverage, in particular for circuits that contain random-pattern resistant faults (circuits with many hard to detect faults). Several techniques have been proposed in order to reduce the test length at the cost of increasing the complexity of the test generator circuit and/or the decoder circuit. In weighted random testing [6, 10] an extra circuit biases the pseudo-random sequence using pre-computed weighted sets. Other techniques try to encode a deterministic test set in the test generator circuit either by searching the appropriate seeds and/or select the LFSR that best covers the test set [5, 7, 9].

Recent work in BIST [4, 2] has led to a new procedure for the generation of test generator circuits which target the minimization of the number of required flip-flops. This procedure is based on the compression of the width (i.e. the number of bits) of the original test patterns. Therefore, a smaller counter-based FSM is used to generate the compressed test patterns (with a width of w bits) which are then fed to the decoding logic. The test pattern generated by the decoding logic is then applied to the circuit under test. The main advantage of this technique is that at the same time we are reducing the area of test generator circuit, we are also cutting down the test time, without introducing additional logic in the decoder circuit.

Width compression was proposed by Chen and Gupta [2] who observed that in many cases, some inputs of the CUT can be connected to the same output of the test generator circuit, without introducing redundant faults, thus not reducing the fault coverage of the circuit. This way, we can reduce the width w of the test generator circuit without introducing, at the same time, extra logic in the decoder circuit. Under these conditions the decoder circuit consists only of interconnecting lines, without any area penalty. Consider the C17 ISCAS'85 [14] benchmark circuit (see Figure 2-a) which can be tested using the set of vectors shown on Figure 2-b. Note that for each test vector, the inputs x_2 and x_6 always assume equal values, therefore they can be driven by the same output of the test generator circuit, as shown in Figure 2-c. Inputs with this characteristic are called *directly compatible.* Inputs x_3 and x_7 are also directly compatible. Observe that input x_1 is always the complement of input x_2, thus they can be derived one from the other using an inverter. This inverter does not increase the complexity of test generator circuit if we consider that we are using flip-flops with inverted and non-inverted outputs. Inputs exhibiting this relationship are referred to as *inversely compatible.* As shown in Figure 2-d this type of compatibility reduced one bit in the test generator circuit width. Other types of compatibility are possible between the inputs but in general they require some logic in the decoder circuit which will introduce some area overhead [4, 13].

Using pre-computed test patterns with don't cares, can significantly simplify the test generator circuit. Test sets with don't cares are in general larger but the number of compatibility inputs may be greater because don't care bits can be chosen to force some type of compatibility, reducing the test generator circuit width.

In this paper we only consider direct and inverse compatibilities on the situations where test patterns exhibit don't cares: the objective will be to minimize the width w of the test generator circuit such that, all non-redundant circuit faults are detected.

3. TEST GENERATION WITH UNSPECIFIED VARIABLE ASSIGNMENTS

In the remainder of the paper, we use the definitions for Satisfiability (SAT) and SAT-based Test Pattern Generation proposed in [11, 15, 12, 16]. A formal model for test generation using SAT models can be found in [12]. Moreover, we briefly review the generation of test patterns with don't cares. In order to maximize compatibility classes, test patterns are required to exhibit don't cares, and thus we base our approach in the test generation model of [17].

x	(x^1, x^0)
0	$(0,1)$
1	$(1,0)$
X	$(0,0)$

Table 1: New variables for modeling unspecified assignments

Given a circuit and its associated CNF formula or a fault f and its associated fault detection formula, the existence 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. (Observe that since only 3^M assignments need to be considered for M variables, the actually required number of Boolean variables is $\lceil \log(3^M) \rceil$, since there are only three possible assignments to each of the original variables. Nevertheless, considering instead $2 \cdot M$ 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 Table 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,

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

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, 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 $(x^0, x^1) = UAND(w_1^0, w_1^1, \ldots, w_j^0, w_j^1)$, and which allows unspecified assignments on the gate inputs and output. Since the simultaneous assignment of any pair of variables (x^0, x^1) to 1 is prevented by (1), 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 $x^1 = AND(w_1^1, \ldots, w_j^1)$. 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 $x^0 = OR(w_1^0, \ldots, w_j^0)$. As a result we obtain from [11, 12],

$$\begin{aligned} \varphi_{u,x^0} &= \left[\prod_{i=1}^{j} (\neg w_i^0 + x^0)\right] \cdot \left(\sum_{i=1}^{j} w_i^0 + \neg x^0\right) \\ \varphi_{u,x^1} &= \left[\prod_{i=1}^{j} (w_i^1 + \neg x^1)\right] \cdot \left(\sum_{i=1}^{j} \neg w_i^1 + x^1\right) \end{aligned} \tag{2}$$

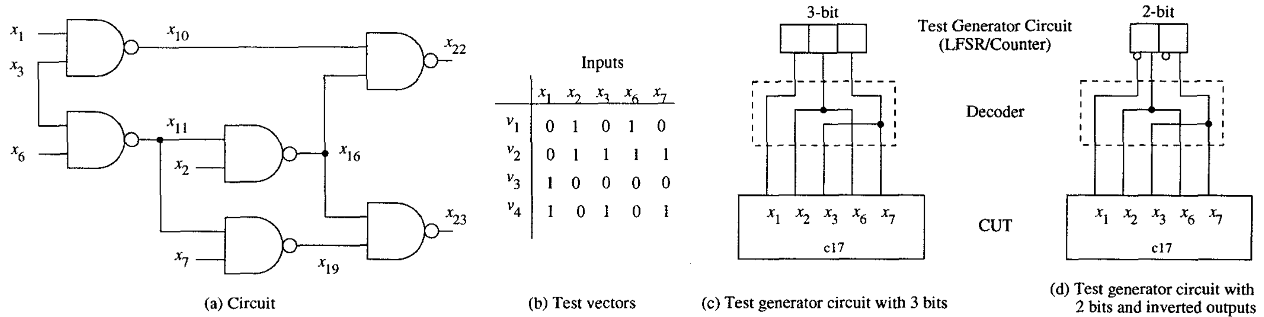

Figure 2: Test set and two BIST generators for C17 circuit[4, 2]

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

$$\varphi_{u,x} = \varphi_{u,x^1} \cup \varphi_{u,x^0} \cup \varphi_{inv,x} \tag{3}$$

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 [12] can be generalized by following the same approach used for deriving (2) [17]. As a result, we can now create the CNF formula for the circuit, one in which unspecified variable assignments are allowed.

4. COMPUTING TEST PATTERNS FOR WIDTH COMPRESSION

In this section we develop a greedy optimization model for computing test patterns aiming at the reduction of the width of the test set. This optimization model is based on test pattern generation in the presence of incompletely specified primary input assignments.

4.1. Forcing Compatibility Classes

A compatibility class is defined as a set of inputs which can be connected together, or through an inverter, such that each of which can be driven by only one output of the test generator circuit throughout the application of the test sequence. Formally we can define directly and inversely compatibility classes as follows:

Definition 1 *Consider a set of test vectors $V = \{T_1, T_2, \ldots, T_N\}$ for a circuit. Two inputs x_i and x_j are **directly compatible** if*

$$\forall T_p \in V \quad T_p[i] = T_p[j] \ \ or \ \ T_p[i] = X \ \ or \ \ T_p[j] = X$$

*and are **inversely compatible** if*

$$\forall T_p \in V \quad T_p[i] = \overline{T_p[j]} \ \ or \ \ T_p[i] = X \ \ or \ \ T_p[j] = X$$

*Two inputs are **compatible** if they are either directly or inversely compatible.*

Definition 2 *The compatibility class set, Ω, is the set of all pairs of inputs that, for a given group of test vectors, are compatible (directly or inversely).*

Inputs		$C_{i,j}$		$U_{i,j}$	$S_{i,j}$
x_i	x_j	Direct	Inverse		
X	X	0	0	0	0
X	0	0	0	1	0
X	1	0	0	1	0
0	X	0	0	1	0
0	0	0	1	1	1
0	1	1	0	1	1
1	X	0	0	1	0
1	0	1	0	1	1
1	1	0	1	1	1

Table 2: Definition of variables $C_{i,j}$, $U_{i,j}$ and $S_{i,j}$

For a set of test vectors which have test patterns with unspecified inputs, finding the optimal set of compatibility classes is an NP-hard problem [4]. So, we use an heuristic approach to efficiently generate good classes in practice. In [4] is presented an heuristic algorithm that produces a good set of compatibility classes. We will use this algorithm as our general procedure to identify compatibility classes. Basically this algorithm identifies directly and inversely compatible inputs, for a given test set, and represents them on a graph whose vertices are the inputs and edges correspond to those compatibility relations between the inputs. Then, heuristically, the procedure identifies an initial set of cliques with size 3, which can be identified in polynomial time [4]. Finally, vertices are added to the cliques if the compatibility is maintained between all the inputs in the clique.

When determining a test vector for a given target fault we would like to keep the same cardinality on the set of compatibility classes already found from previous test vectors. Thus we will associate a compatibility boolean variable, $C_{i,j}$, with each pair of inputs that are in a compatibility class. As shown in Table 2 this variable assumes the value 1 when the compatibility between inputs x_i and x_j does not hold. Considering that each circuit node x is represented by two variables in the model, x^0 and x^1 according to Table 1, the clauses $\varphi^{C_{i,j}}$ which define this variable for direct and inverse compatibility are the following:

$$\varphi^{C_{i,j}} = \begin{cases} (\neg x_i^1 + \neg x_j^0 + C_{i,j}) \cdot & \text{if } x_i \text{ and } x_j \text{ are} \\ (\neg x_i^0 + \neg x_j^1 + C_{i,j}) & \text{directly compatible} \\ & \\ (\neg x_i^1 + \neg x_j^1 + C_{i,j}) \cdot & \text{if } x_i \text{ and } x_j \text{ are} \\ (\neg x_i^0 + \neg x_j^0 + C_{i,j}) & \text{inversely compatible} \end{cases} \tag{4}$$

Moreover, in order to enhance future compatibility classes we add to our model two more types of boolean variables. The unity variables, $U_{i,j}$, which allow us to identify when any of the inputs in a compatibility class pair is specified, and the simultaneity variables, $S_{i,j}$, that determine when both of the inputs are specified. With this variables we are able to increase the number of unspecified inputs in each test pattern within a compatibility class. Table 2 describes the values assigned to these variables for any valid combination of two input bits. The clauses that capture this behavior are:

$$\varphi^{U_{i,j}} = (\neg x_i^0 + U_{i,j}) \cdot (\neg x_i^1 + U_{i,j}) \cdot (\neg x_j^0 + U_{i,j}) \cdot (\neg x_j^1 + U_{i,j}) \quad (5)$$

$$\begin{aligned} \varphi^{S_{i,j}} &= (\neg x_i^0 + \neg x_j^0 + S_{i,j}) \cdot (\neg x_i^0 + \neg x_j^1 + S_{i,j}) \cdot \\ &\quad (\neg x_i^1 + \neg x_j^0 + S_{i,j}) \cdot (\neg x_i^1 + \neg x_j^1 + S_{i,j}) \end{aligned} \quad (6)$$

4.2. The Complete Optimization Model

The main objective in the computation of test patterns for width compression is to identify a test pattern that detects a fault but, if possible, does not increase the number of compatibility classes defined by the test vectors previously found. Hence, our goal is to minimize the number of extra compatibility classes such that the given fault is still detected. Additionally, we also would like to increase the number of unspecified inputs. This way, we increase the probability that our class compatibility identification procedure will find in the future other set of compatibility classes with the same cardinality, even if this particular vector increases the number of compatibility classes. As a result we obtain the following optimization model,

$$\begin{aligned} &\text{minimize} \quad \sum_{(i,j)\in\Omega} (C_{i,j} + U_{i,j} + S_{i,j}) \\ &\text{subject to,} \\ &\qquad \bigcup_{(i,j)\in\Omega} (\varphi^{C_{i,j}} \cdot \varphi^{U_{i,j}} \cdot \varphi^{S_{i,j}}) \\ &\qquad \varphi_u^D \end{aligned} \quad (7)$$

(where φ_u^D denotes the set of clauses for detecting a fault under unspecified primary input assignments [17].) This formulation basically requires that the total number of compatibility classes and assigned input variables be minimized under the constraint that the fault be detected. Note that according to Table 2, each item for a pair of inputs in the cost function assumes only four discrete values: 0, if both inputs are unspecified; 1 if only one of the inputs is specified; 2 if both inputs are specified but the compatibility is maintained; 3 otherwise.

Given the mapping between CNF clauses and linear inequalities described in [18], (7) can be viewed as an integer linear program, and so different integer linear optimization packages can potentially be applied. Nevertheless, the constraints of (7) are tightly related with propositional satisfiability. Consequently, and as shown in [18], SAT based ILP solvers are preferable for solving ILPs for which the constraints correspond to CNF formulas.

Notice that, when there are no compatibility classes ($\Omega = \emptyset$) none of the variables for compatibility, unity and simultaneity are defined. In that case, the cost function to minimize is redefined to minimize the number of specified inputs in the test pattern, as presented in [17]:

$$\text{minimize} \quad \sum_{x_i \in PI} (x_i^0 + x_i^1) \quad (8)$$

Circuit	#PI	ATALANTA		MTP-C		#bits	Sec/
		#V	W	#V	W	red.	Vect.
9symml	9	94	9	84	9	0	1.5
alu4	14	239	14	197	12	2	12.4
cht	47	194	5	178	4	1	2.3
cm138a	6	13	6	12	6	0	0.1
cm150a	21	65	10	44	7	3	4.3
cm163a	16	44	8	35	8	0	0.2
cmb	16	39	12	30	12	0	0.1
comp	32	68	32	50	20	12	15.5
comp16	35	111	33	94	28	5	18.8
cordic	23	59	21	45	17	4	7.7
cu	14	49	13	36	10	3	0.1
dalu	75	740	26	601	25	1	33.3
majority	5	11	5	11	5	0	0.1
misex1	8	28	5	22	5	0	0.1
misex2	25	77	14	65	14	0	0.4
misex3	14	289	14	241	14	0	13.0
mux	21	64	10	43	7	3	2.0
pcle	19	75	11	53	11	0	0.2
pcler8	27	90	12	74	12	0	0.4
term1	34	135	17	102	16	1	4.4
too_large	38	226	31	162	31	0	12.1
unreg	36	133	5	122	5	0	3.2

Table 3: Experimental results for IWLS'89 benchmarks

5. EXPERIMENTAL RESULTS

The model described in the previous section has been integrated in a test pattern generation framework for the computation of test patterns referred to as *Minimum Test Pattern generator with Width Compression* (MTP-C), which uses the SAT-based ILP algorithm of *bsolo* [18] and the fault simulator provided with ATALANTA [19]. The results included below were obtained with the IWLS'89 benchmark suite [20] and with the ISCAS'85 benchmark suite [14]. In all cases MTP-C was run with a bound on the amount of allowed search (i.e. the total number of conflicts was limited to 1000). This permits MTP-C to identify acceptable solutions, which in some cases may not be necessarily optimal. Moreover, in order to facilitate the optimization process, MTP-C 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 [12] is used for computing a startup test pattern.

Table 3 contains the results for the IWLS'89 benchmarks for both ATALANTA [19], and MTP-C. ATALANTA is an ATPG tool that can generate test patterns with don't cares. Columns #PI, #V and W denote, respectively, the number of primary inputs, number of test vectors and with of test set after compression. Last two columns represent the number of bits gained by our ATPG over ATALANTA and the average time to solve the model for each vector (on a Sun Sparc-Ultra with 384 Meg. of memory).

From these results we verified that the model implement by the MTP-C ATPG decreases the number of bits necessary to fully test a circuit. This is achieved by computing fewer test vectors than ATALANTA and, in some cases, reducing the width, in number of bits, necessary to store the test patterns. We should notice that a

Circuit	#PI	ATALANTA		MTP-C		#bits red.	Sec/ Vect.
		#V	W	#V	W		
c432	36	148	31	107	24	7	25.7
c499	41	103	41	66	40	1	42.4
c880	60	393	30	289	29	1	28.2
c1355	41	145	41	96	37	3	77.9
c1908	33	291	31	181	29	2	46.3
c2670	233	780	55	600	52	3	91.0
c3540	50	718	30	530	31	-1	98.4
c5315	178	1393	38	993	44	-6	224.7
c6288	32	248	32	58	32	0	326.3
c7552	207	919	101	706	93	8	166.8

Table 4: Experimental results for ISCAS'85 benchmarks

reduction by one bit in the width of the test patterns will reduce to half the number of test patterns produced by the BIST generator, thus will cut down to half the time required to fully test the circuit.

Table 4 contains the results for the ISCAS'85 circuits. Once more we can conclude that MTP-C is able to improve over the ATALANTA results, but in this case the improvements are in general higher. This may happen because now we are dealing with bigger circuits, so the number of distinct vectors that can detect a given fault may be bigger, therefore it will be "easier" for the ILP solver to find vectors that meet the compatibility classes restrictions.

The results obtained by circuits c3540 and c5315 shows that our greedy approach may not produce good results for all circuits. Selecting a good fault ordering and improving the compatibility selection procedure between two faults are techniques that should be considered to further improvement.

From the previous experimental results for the IWLS'89 and ISCAS'85 benchmarks we can conclude that for some circuits our approach can reduce the test patterns width, which has a significant impact on the test time and area of test generator circuit. The heuristics incorporated in the MTP-C ATPG tool need to be tuned in order to achieve a better width compression in a wide range of circuits. We should also note that counter-based test generation circuits are only practical provided we are able to reduce the width to no more than 25-30 bits. Hence, for several of the benchmarks, additional width compression is required.

6. CONCLUSIONS

In this paper we introduce a SAT-based integer linear programming model for computing test patterns for width compression. Based on this model we describe an ATPG tool (MTP-C) which incorporates several heuristics. The applicability of the model is illustrated by computing test patterns for several benchmark circuits. The next step of this work is to determine new types of compatibility (such as d-compatible [4] and others) to be incorporated in the model with the objective of reducing the test patterns width without a significantly increase in the area of the decoder.

Additional research work involves further constraining the ILP formulation so that larger problem instances can be solved optimally. Furthermore, a good set of heuristics to determine the compatibility sets and define the sequence of the sequence of the faults has to be developed. Finally, a long-term objective of this work is the integration of the proposed model in a complete testing environment, enabling the use of compressed test patterns for the synthesis of reduced-size FSMs for BIST in specific target applications.

7. REFERENCES

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

[2] C.-A. Chen and S. K. Gupta, "A methodology to design efficient BIST test pattern generators," in *Proceedings of International Test Conference*, pp. 814–823, 1995.

[3] C. Dufaza and G. Canbom, "LFSR based deterministic and pseudo-random test pattern generator structures," in *Proceeding of European Test Conference*, pp. 27–34, 1991.

[4] K. Chakrabarty, B. T. Murray, J. Liu, and M. Zhu, "Test width compression for built-in self testing," in *Proceedings of International Test Conference*, November 1997.

[5] S. Hellebrand, B. Reeb, S. Tarnick, and H.-J. Wunderlich, "Pattern generation for deterministic BIST scheme," in *Proceedings of International Conference on Computer-Aided Design*, 1995.

[6] F. Muradali, V. Agarwal, and B. Nadeau-Dostie, "A new procedure for weighted random built-in self-test," in *Proceedings of International Test Conference*, pp. 660–669, 1990.

[7] I. Pomeranz and S. M. Reddy, "A learning-based method to match a test pattern generator," in *Proceedings of International Test Conference*, pp. 998–1007, 1993.

[8] N. A. Touba and E. J. McCluskey, "Synthesis of mapping logic for generating transformed pseudo-random patterns for BIST," in *Proceedings of International Test Conference*, pp. 674–682, 1995.

[9] S. Venkataraman, J. Rajski, S. Hellebrand, and S. Tarnick, "An efficient BIST scheme based on reseeding of multiple polynomial linear feedback shift registers," in *Proceedings of International Conference on Computer-Aided Design*, pp. 572–577, 1993.

[10] H. Wunderlich, "Multiple distribution for biased random test patterns," in *Proceedings of International Test Conference*, pp. 236–244, 1998.

[11] T. Larrabee, "Test pattern generation using boolean satisfiability," *IEEE Transactions on Computer-Aided Design*, vol. 11, pp. 4–15, January 1992.

[12] J. P. Marques-Silva and K. A. Sakallah, "Robust search algorithms for test pattern generation,," in *Proceedings of Fault-Tolerant Computing Symposium*, June 1997.

[13] K. Chakrabarty and B. T. Murray, "Design of built-in test generator circuits using width compression," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 17, pp. 1044–1051, October 1998.

[14] F. Brglez and H. Fujiwara, "A neutral list of 10 combinational benchmark circuits and a target translator in FORTRAN," in *Proceedings of International Symposium on Circuits and Systems*, 1985.

[15] S. T. Chakradhar, V. D. Agrawal, and S. G. Rothweiler, "A transitive closure algorithm for test generation," *IEEE Transactions on Computer-Aided Design*, vol. 12, pp. 1015–1028, July 1993.

[16] P. R. Stephan, R. K. Brayton, and A. L. Sangiovanni-Vincentelli, "Combinational test generation using satisfiability," *IEEE Transactions on Computer-Aided Design*, vol. 15, pp. 1167–1176, September 1996.

[17] P. F. Flores, H. C. Neto, and J. P. M. Silva, "An exact solution to the minimum-size test pattern problem," in *Proceedings of International Conference on Computer Design*, October 1998.

[18] V. Manquinho, P. Flores, J. Marques-Silva, and A. L. Oliveira, "Prime implicant computation using satisfiability algorithms," in *Proceedings of International Conference on Tools with Artificial Intelligence*, November 1997.

[19] H. K. Lee and D. S. Ha, "On the generation of test patterns for combinational circuits," Tech. Rep. 12_93, Department of Electrical Engineering, Virginia Polytechnic Institute and State University, 1993.

[20] "Test benchmark suite." International Workshop on Logic Synthesis 1989, Available from http://www.cbl.ncsu.edu/pub/Benchmark_dirs/LGSynth89/, 1989.