LOT: Logic Optimization with Testability - New Transformations using Recursive Learning *

Mitrajit Chatterjee Dhiraj K. Pradhan Wolfgang Kunz
Laboratory for Computer and Digital Systems
Department of Computer Science
Texas A & M University
College Station, TX 77843.

Abstract: *A new approach to optimize multi-level logic circuits is introduced. Given a multi-level circuit, the synthesis method optimizes its area, simultaneously enhancing its random pattern testability. The method is based on structural transformations at the gate level. New transformations involving EX-OR gates derived based on indirect implications by Recursive Learning have been introduced in the synthesis of multi-level circuits. This method is augmented with transformations that specifically enhance random-pattern testability while reducing the area. Testability enhancement is an integral part of our synthesis methodology. Experimental results show that the proposed methodology can not only realize lower area, but also achieves better testability compared to testability enhancement synthesis tools such as* tstfx. *Specifically for ISCAS-85 benchmark circuits, it was observed that EX-OR gate-based transformations can yield smaller circuits compared to state-of-the-art logic optimization tools like SIS and HANNIBAL.*

1 Introduction

Traditional goals in all automatic synthesis and optimization of multi-level combinational circuits [10, 8, 6, 4, 3, 5, 11] include low area, high performance and power. In almost all cases, synthesis of multi-level circuits only concentrates on one of its goals and ignores other design requirements. With the growing demand in effective automatic synthesis methods, targetting multiple design requirements is of practical importance. Although EX-OR gate-based synthesis [18] of two or three-level logic has been explored, no work has yet been reported on multi-level logic synthesis using EX-OR gates. Efficient usage of EX-OR gates in multi-level circuits can be advantageous to a variety of circuits. These EX-OR gates can not only reduce the circuit area but also, they make the circuit more easily testable [20, 22]. In this paper, synthesis is done with reference to *area optimization* and *random pattern testability enhancement.* Transformations done during synthesis include new transformations introducing EX-OR gates in multi-level circuits. Our synthesis method, Logic Optimization with Testability (LOT), can target area optimization as well as random pattern testability. Also, for the first time, a uniform framework is provided able to integrate multiple design goals in a single tool. It is important to note that transformations using indirect implications [1, 3] provide a powerful, unified theory for the entire synthesis process. Further extensions of this framework to delay optimization is currently underway.

The synthesis procedure is based on the gate-level description of a multi-level circuit. It has been observed that the logic optimization with Boolean transformations at the structural level [4, 9, 5, 3] can be more memory-efficient [3] and closer to the physical reality of the design compared to a functional level logic optimization based on Boolean networks or BDDs [10, 11]. Furthermore, working on the gate-level can give a better view of other design costs during synthesis. Our synthesis method applies logic transformations to a multi-level circuit to yield reduced area as well enhances random testability. The transformations used here are of two types:
(a) Transformations based on indirect implications derived by recursive learning [3, 1].
(b) New transformations using EX-OR gates.
The transformations of the first type make use of efficient redundancy identification techniques to make a wide range of circuit transformations. These transformations can perform arbitrary manipulations in a combinational network [3], thus covering a large design space. The transformations of the second type are based on introduction of EX-OR gates in the circuit. Using EX-OR gates as primitive gates in digital circuits has drawn attention in synthesis [18, 19, 22], particularly in implementing arithmetic and linear functions, telecommunication, encryption and encoding schemes [18]. Recent designs of PLD and FPGAs are also including EX-OR gates in their logic units. Moreover, circuits with more EX-OR gates tend to be more testable [20, 22]. Transformations of both types can

*Research reported is supported partly by NSF grant MIP 9406946 and ONR grant # N00014-92-J-1366. W. Kunz has a joint appointment with Texas A&M University and Max-Planck-Society, Fault Tolerant Computing Group, Potsdam, Germany. This work is a collaborative research effort between the two instituitions.

be used iteratively to yield an optimized circuit.

While most of these tools ensure that all faults of interest are testable, *random testability* [21] of the faults, used in the context of built-in-self-test (BIST), is not considered. Sythesis techniques presented to date for random pattern testable circuit required two-level circuit description as input [12, 13, 14]. Our technique presented here differs in that it can take any multi-level circuit as input to the synthesis tool. Therefore, it is more versatile in the environment of engineering changes. In addition, our tool relies on the unifying theory of transformations through indirect implications [1, 3] to achieve both area optimization and testability.

The paper is organized as follows. The next Section provides a brief summary of previous work. Section 3 outlines some preliminaries and reviews logic synthesis using Implication-based Boolean (IB) Transformations. Section 4 introduces new transformations based on EX-OR gates. Section 5 presents synthesis for area optimization and its performance in ISCAS-85 benchmark circuits. Section 6 describes an algorithm for random pattern testability enhancement presenting related experimental results. We conclude in Section 7.

2 Previous Work

A survey of multi-level combinational logic optimization techniques based on Boolean networks can be found in [11]. Logic synthesis based on structural transformations has been effectively applied in [4, 9, 3, 5]. Synthesis for area optimization was done by transformations [4, 5] using redundancy elimination based on adding and removing connections in the circuit. These transformations were also applied in the form of permissible bridges after technology mapping in [9]. A generalized form of the IB transformations, based on orthonormal expansions, have been presented in [3].

Logic synthesis, with extensive use of EX-OR gates, has been limited to two-level (AND-EXOR) and three-level (AND-OR-EXOR) designs (see [18]). These methods develop algorithms for optimizing exclusive-or sum-of-products yielding a minimum number of product terms. Earlier, [20, 22] have shown that EX-OR gates enhance testability for AND-EXOR designs.

Logic synthesis with testability enhancement has been studied at various levels of synthesis - starting from high-level synthesis to technology mapping [17]. In the context of random testable logic synthesis of the unmapped circuit, previous research consists of redundancy removal from combinational circuits [11], synthesis of fully testable circuits [8, 15], testability-preserving of multiple stuck-at-faults, path delay faults [17] and synthesis of random testable circuits from two-level circuits [13, 14].

It has been proposed in [12] that careful assignment of don't cares of functions and redundant lines can improve the detectability profile of circuits. The two previous random pattern testable circuit synthesis methods [13, 14] start with a two-level circuit, first determing the random testability of the circuit and identifying the hard faults. Multi-level circuits are synthesized from two-level circuits using transformations, each step being evaluated based on its impact on the random testability of the circuits.The transformations in [13] are limited to algebraic factors, namely kernels and common cubes. The algebraic transformations used in [14] include *single cube division*, *double cube division* and some *double cube divisions with multiple outputs*. The synthesis procedure in [13] also uses testability-preserving transformations proposed in [8, 15] and inserts test points based on fault detection probabilities.

3 Transformations for Synthesis

The proposed synthesis method takes as its input a multi-level combinational circuit and optimizes area while enhancing testability. The cost function used, therefore, can include both area and random pattern testability. Structural transformations are used to minimize the cost function. This Section gives the basic preliminaries for synthesis, and presents the transformations to be used here.

3.1 Preliminaries

Assume a combinational circuit, C, is given with n primary inputs and m primary outputs. The combinational circuit consists of primary gates like AND, OR, NOT, NAND, NOR and EX-OR gates. All gates in the circuit have a unique label and their output signals, y_i, realize Boolean functions $y_i(\vec{x})$: $B_2^n \rightarrow B_2$ with $B_2 = \{0,1\}$, where the variable $x_1, ..x_n$ corresponds to the primary input signals of the circuit C. Following the usual representation of a combinational circuit as a directed acyclic graph (DAG), a signal f lies in the *transitive fanout* of y if and only if there exists a directed path from y to f. Furthermore, we assume that there are no *external* don't cares, the function of the combinational network $C(\vec{x})$:$B_2^n \rightarrow B_2^m$ with $B_2 = \{0,1\}$ is completely specified.

Two combinational networks, C and C', are called *equivalent*, denoted $C = C'$, if they implement the same function $C(\vec{x})$:$B_2^n \rightarrow B_2^m$ with $B_2 = \{0,1\}$. A transformation from a network C to another equivalent network C' is possible by replacing the node y in C by a node y' with an equivalent function. A function at node y can also be replaced by some non-equivalent function y' if this does not change the function $C(\vec{x})$:$B_2^n \rightarrow B_2^m$ of the logic network as a whole. Such functions are called *permissible functions* [6].

3.2 Implication-based Boolean Transformations - Review

Indirect implications derived by using Recursive Learning have been shown to be useful in testing verification and optimization[1, 2, 3]. Given a set of value assignments, Recursive Learning can be used to obtain all indirect implications. The procedure allows the user to set the maximum level of recursion to control the computational time. Therefore, this provides a way of trading off between time for transformations and redundancy removal. In this paper, we rely on Boolean transformations derived by using indirect implications. These transformations can be referred here as Implication-based Boolean (IB) transformations. They have three main advantages :

	Condition	*Transformations*
1	y=0 ⇒ f=1	y'= $\bar{f}$ + y$\vert_1$
2	y=0 ⇒ f=0	y'= f + y$\vert_0$
3	y=1 ⇒ f=1	y'= f.y$\vert_1$
4	y=1 ⇒ f=0	y'= $\bar{f}$.y$\vert_0$

Table 1: *Transformations based on Implications*

- They are simple and the cost of each transformation can be estimated with low time overhead.
- These transformations preserve the functionality of the circuit and therefore there is no need to verify for equivalence after transformation unlike [5].
- The expansions based on IB transformations can cover a wide variety of logic transformations [3].

The following reviews the basic transformations already presented in [3]. We enhance these transformations with new ones for our tool. The factorization techniques commonly used in multi-level optimization can be derived based on the expansion:

$$y(\vec{x}) = f(\vec{x}).y(\vec{x}) \mid_{f(\vec{x})=1} + \bar{f}(\vec{x}).y(\vec{x}) \mid_{f(\vec{x})=0} \qquad (1)$$

Here, we use the following notation, $y = f.y \mid_1 + \bar{f}.y \mid_0$ to represent the above equation.
The above expression can take the form of a division operation with $f(\vec{x})$ as the *divisor*, $y(\vec{x})$ as the *quotient*, $y(\vec{x}) \mid_{f(\vec{x})=1}$ as the *dividend* and $\bar{f}(\vec{x}).y(\vec{x}) \mid_{f(\vec{x})=0}$ as the *remainder*. The terms $y(\vec{x}) \mid_{f(\vec{x})=1}$ and $y(\vec{x}) \mid_{f(\vec{x})=0}$ denote the cofactors of this expansion. The above expression can be interpreted as a generalized form of the well-known Shannon expansion [3] The main issue of this approach is to divide [11, 10] function $y(\vec{x})$ by appropriate divisors $f(\vec{x})$ such that the exploitation of the internally created don't cares results in a reduced circuit. Though don't care conditions are not explicitly calculated, a test pattern generation tool is used to remove the redundancies resulting from these don't cares. This leads to the following two-step process :

1. Transformation: $y = f.y + \bar{f}.y$
2. Reduction: Redundancy elimination.

The above can be seen as performing a special way of doing a Boolean division [11] for some dividend y and some divisor f (Boolean division is not unique). The method to identify divisors is based on *indirect implications* [1, 2]. If a value assignment at a node y allows us to imply a unique value assignement at node f, then the four transformations in Table 1 are valid [3]. The node f must *not* be in the transitive fanout of y, to ensure that the circuit remains combinational after the transformation.

Example 3.1: Take the circuit shown in Fig1(a) as an example. It can be observed that $y = 0 \Rightarrow f = 0$ and transformation based on *Condition 2* can be applied on the circuit. The resultant circuit is shown in Fig 1(b). Redundancy elimination on the resultant circuit would determine lines 'c' and 'd' to be as redundant s-a-0 lines and hence, can be eliminated. The final circuit after redundancy elimination is shown in Fig 1(c).□.

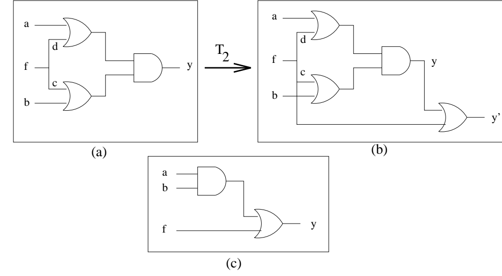


Figure 1: *An example of an IB transformation.*

Transformations identified by 'D-implications' [3] are related to *permissible functions.*
Definition 3.1: If $f = U$, $U \in 0,1$ is a value assignment at node f which is *necessary* to detect the fault, y s-a-V, at least one primary output of the combinational network C, then $y = \bar{V}$ 'D-implies' $f = U$ and is denoted by: $y = \bar{V} \xrightarrow{D} f = U$. The conventional implications can be viewed as a special case of such D-implications. These implications can exploit the controllability as well as the observability conditions in the concerned nodes. The transformations defined in [3] are the following :

Theorem 3.1: (Analogous to *Condition 1*) Let f and y be arbitrary nodes in a combinational network C where f is not in the transitive fanout of y and y is an irredundant node in the network. The function y' with $y' = y \mid_1 + \bar{f}$ is a permissible function at node y iff the D-implication $y = 0 \xrightarrow{D} f = 1$ is true [3].□

Theorem 3.2 - 3.4: Analogous to *Conditions 2 - 4*.

Transformations 1-4: Transformations based on *Theorems 3.1-3.4* followed by redundancy removal.

D-implications to be used in the transformations are determined by Recursive Learning. This is accomplished by two routines: *make_all_implications()* and *fault_propagation_learning()*, as given in [1], if they are performed to the Roth's five-valued logic alphabet. The implication-based synthesis procedure only looks at *indirect implications* as the promising candidates for optimization.

4 Transformations with EX-OR gates

Circuits transformations based on *Transformations 1 - 4* can make arbitrary manipulations in a combinational network based on AND, OR and NOT gates. It was observed that transformations involving EX-OR gates can provide an added direction in the search for circuit optimization. New implication-based transformations introduce EX-OR gates allowing (a) replacing a 2-input gate with an EX-OR or EX-NOR gate and (b) introducing an extra EX-OR gate. It may be noted that, as in [3], these transformations are guaranteed to preserve functionality. Also they introduce redundancy in the circuit which, when removed, can often lead to a reduced circuit.

4.1 Transforming Gate Functionality

Two-input gates in circuits can be transformed to EX-OR gates or EX-NOR gates if they are *permissible*, based on implications [6, 5, 11]. However, a two-input gate, when transformed to an EX-OR gate, does

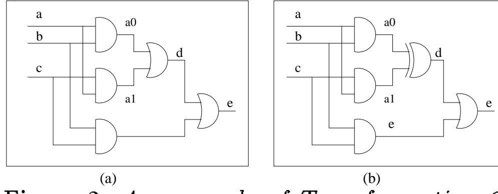


Figure 2: *An example of Transformation 6.*

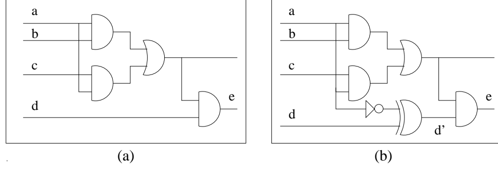


Figure 3: *An example of Transformation 8.*

not necessarily enhance the random testability. If the transformation is followed by some other transformations, as we show, the EX-OR gate may play an active role in random testability enhancement. A two-input AND (OR) gate or a two-input NOR (NAND) gate can be transformed to an EX-NOR (EX-OR) gate due to an observability don't care condition [11, 5]. We can identify two such types of gate transformations. The proofs the theorems can be found in [7].

Theorem 4.1: Let y be a node feeding from an AND (NOR) gate with two inputs, a_0 and a_1. The condition $y = 0 \xrightarrow{D} (a_0 \oplus a_1 = 1)$ is said to exist if a change in node y from 0 to 1 can only be observed at the primary outputs if $(a_0 \oplus a_1 = 1)$ and the AND (NOR) gate can be replaced by a *permissible* EX-NOR gate.□

Theorem 4.2: Let y be a node feeding from an OR (NAND) gate with two inputs, a_0 and a_1. The condition, $y = 1 \xrightarrow{D} (a_0 \oplus a_1 = 1)$ is said to exist if a change in node y from 1 to 0 can only be observed at the primary outputs if $(a_0 \oplus a_1 = 1)$ and the OR (NAND) gate can be replaced by a *permissible* EX-OR gate.□

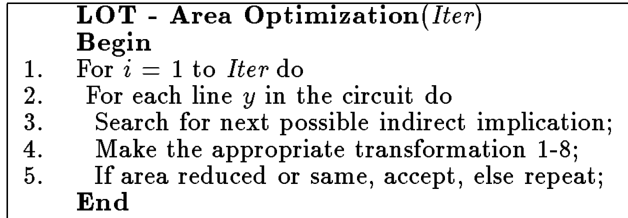
Transformation 5-6: Transformations based on *Theorems 4.1-4.2* followed by redundancy removal.

Example 4.1: The circuit shown in Fig.2(a) can be represented as $(ab+bc+ca)$. At node d, the condition for a change from 1 to 0, to be observed at e, is $\{(a_1,a_0)=(0,1),(a_1,a_0)=(1,0)\}$. This condition can allow the two-input OR gate at node d to be transformed to an EX-OR gate, (Transformation 6), yielding the circuit in Fig.2(b).□

The search for such gate functionality transformations can be used iteratively between the other IB transformations.

4.2 Transforming with EX-OR Gates Based on D-Implication

In the above, we described how one can replace a two-input gate with an EX-OR gate. The following shows how one can combine lines through an additional EX-OR gate like it is done in Transformations 1-4. EX-OR gates can be introduced as *permissible* functions based on conditions testing for line y stuck-at-0 as well as the line y stuck-at-1. The following transformations give the condition for introducing *permissible* EX-OR gates.



LOT - Area Optimization(*Iter*)
Begin
1. For $i = 1$ to *Iter* do
2. For each line y in the circuit do
3. Search for next possible indirect implication;
4. Make the appropriate transformation 1-8;
5. If area reduced or same, accept, else repeat;

End

Figure 4: Algorithm LOT for Area Optimization.

Theorem 4.3: Let f and y be arbitrary nodes in a combinational network, C, where f is not in the transitive fanout of y and y is an irredundant node in the circuit. The function y' with $y' = y \mid_1 \oplus f$ is a permissible function at node y iff the D-implications $y = 0 \xrightarrow{D} f = 0$ and $y = 1 \xrightarrow{D} f = 0$ are true.□

Theorem 4.4: Let f and y be arbitrary nodes in a combinational network, C, where f is not in the transitive fanout of y and y is an irredundant node in the circuit. The function y' with $y' = y \mid_1 \oplus \bar{f}$ is a permissible function at node y iff the D-implications $y = 0 \xrightarrow{D} f = 1$ and $y = 1 \xrightarrow{D} f = 1$ are true.□

Transformation 7-8: Transformations based on *Theorems 4.3-4.4* followed by redundancy removal.

Example 4.2: The circuit shown in Fig.3 illustrates an example for *Transformation 8*. The conditions required to test d stuck-at 0 as well as d stuck-at 1 both include the assignment $(a = 1)$. Thus, from the definition of D-implications, we get $d = 0 \xrightarrow{D} a = 1$ and $d = 1 \xrightarrow{D} a = 1$ as true. The resultant circuit is shown in Fig.3(b).□

The transformations introduced here are new and can be used for area optimization as well as for random testability enhancement in combinational circuits. Introduction of the EX-OR gate may result in more than one line becoming redundant, thus aiding area optimization. The EX-OR gate will also enhance the observability of all the faults in the input cone of, f, resulting in better random testability.

5 Synthesis for Area Optimization

Here, the main emphasis during optimization is on area only. The new transformations introduced in the paper are used in combination with the existing IB transformations [3] to get a stronger combinational multi-level area optimization tool. Application of EX-OR based transformations is motivated by the fact that optimized AND-EXOR expressions require fewer products than sum-of-product expressions in many circuits [18]. Efficient library implementation of EX-OR gates can ensure that their extra delay and area overhead is not significant compared to other gates and routing. EX-OR gates can be particularly useful if they can replace several two-input gates. Transformations are applied on the circuits based on *Transformations 1 - 8* and are accepted based on the resultant area of the circuit. This method is greedy in nature and accepts any of the transformations which can reduce the area. The new transformations based on *Transformations 5 - 8* can cover new search spaces during optimization and thus have the potential to

Circuit		Original	SIS 1.1	HANNIBAL	LOT	Time	Comparison	
No.	Name	2-i/p GE	2-i/p GE	2-i/p GE	2-i/p GE	hr:min	LOT/SIS	LOT/HANN
1	c432	175	123	104	105	00:05	0.85	1.01
2	c499	340	384	333	328	00:01	0.85	0.98
3	c880	320	300	291	286	00:03	0.95	0.98
4	c1355	496	384	379	328	00:03	0.85	0.86
5	c1908	530	357	356	328	00:15	0.92	0.92
6	c2670	780	533	516	520	00:20	0.97	1.01
7	c3540	1113	909	863	835	02:20	0.91	0.96
8	c5315	1746	1282	1364	1330	01:39	1.03	0.97
9	c6288	2384	2187	1871	1864	06:10	0.85	0.99
10	c7552	2367	1694	1352	1367	10:11	0.80	1.01

Table 2: *Comparison of Area with SIS and HANNIBAL.*

Circuit Name	2-i/p EX-OR Gates		Total # of 2-i/p Gates
	Original	LOT	
c432	18	0	105
c499	104	104	172
c880	0	27	245
c1355	0	104	172
c1908	0	63	234
c2670	0	52	442
c3540	0	131	780
c6288	0	15	1841
c7552	0	283	943

Table 3: Estimate of EX-OR gates.

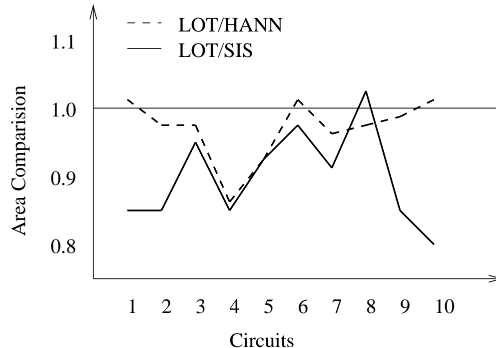


Figure 5: *Area Savings Compared to SIS & HANNIBAL.*

create a smaller circuit. The procedure stops after a pre-determined number of iterations across the circuit.

Experimental Results

Experiments were conducted on the ISCAS 85 benchmark circuits to demonstrate the combined strength of the transformations. Our tool, LOT, has been implemented by making extensions to the HANNIBAL code [3] which also includes the public domain fault simulator FSIM. A short pseudo-code of the implementation has been presented in Fig. 4. The results, presented in Table 2, show that the new EX-OR-based transformations can make a difference in the area of the circuits. During area estimation, technology mapping would have given the best estimate of the circuit area. However, we have used the number of *2-input gates*, denoted as 2-i/p GE (equivalent to half the *number of connections* in [3]) to evaluate the resultant area. Though a two-input EX-OR gate can be counted as 2 two-input gates [18], here, a two-input EX-OR gate is counted as 2.5 two-input gates (as in [23]). This makes the comparison fairer. The circuits synthesized by the proposed tool, LOT, were compared with that synthesized by SIS 1.1 (*script.rugged*) and HANNIBAL [3]. It may be seen that there is a potential for smaller area by LOT when compared to SIS and HANNIBAL. It is interesting to note that the amount of savings for LOT over SIS has a similar pattern to that over HANNIBAL, as shown in Fig. 5. The synthesis time, as reported in the 7th column, was measured in a Sparc 5 machine and is comparable to [3]. The run-time can be much improved by a better software implementation of the tool. One of the areas we can obtain savings in time is in redundany removal by using a better test generation tool. One other tradeoff that needs to be studied is increasing the maximum level of Recursion Learning which can yield more indirect implications, and therefore, more flexibility in the search for optimality.

Table 3 presents the number of EX-OR gates that were used in the optimized circuits. The number of two-input EX-OR gates has been presented in column 3. Column 4 gives the total number of two-input equivalent gates in the circuit (as has been measured in [5]). Here a 2-input EX-OR gate is counted as a 2-input gate. As can be observed, LOT incorporates many EX-OR gates in the optimized circuit and uses them in the circuit. As an example, it could identify all the EX-OR gate structures in c1355 and the resultant structure is now similar to c499 (c1355 was derived from c499 by transforming its XOR gates to NAND gate representations).

6 Random Pattern Testability Enhancement

The following describes the integrated framework for the tool to simultaneously optimize area and enhance testability. The underlying philosophy here is to integrate the primary objective of optimal area with a secondary objective of testability. However, the proposed framework allows for reordering and expanding the objectives. For example, if the primary objective is testability and the secondary objective, area, the ordering of the optimization procedures can be changed as shown here.

Effective BIST requires design of random pattern

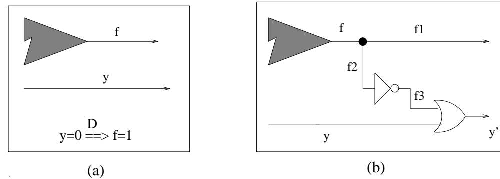


Figure 6: *Transformation 1 - an example.*

testable circuits or design of test pattern generators (TPGs) tailored for the circuit under test (CUT). Those faults that need long TPG sequences for testing are termed as *hard-to-detect* (HTD) faults. A less randomly testable circuit will have a high number of (HTD) faults. Standard techniques used to test HTD faults, using shorter TPG sequences, use complex additional hardware, either inside the circuit with additional pins (test points) or in the TPG[21]. The additional hardware incurred can be miminized if the circuits are synthesized with a higher random pattern testability.

During synthesis, most of the transformations applied here would result in reducing the area - but some may increase the area to sacrifice for better testability. The two main features include a synthesis guidance procedure based on random pattern testability and a method for efficient detection of the cost function of the circuit. The cost function here will depend on the fault detection probabilities in the circuit and an approximate estimate is used to guide the search.

6.1 Calculating Detection Probabilities

As the proposed synthesis procedure is guided by the random testability of the combinational circuit at each iteration, the approximate estimation of the detection probabilities of the faults should have low time overhead. Our synthesis method is based on statistical estimation of fault detection probabilities, as proposed in STAFAN [16]. STAFAN uses results on controllability of lines, as well as statistics on sensitization frequency of lines computed by simulating a given set of patterns on the circuit, to deduce fault detection probabilities. Simulations are done to estimate the controllabilities and path sensitization probabilities at each line in the circuit. The method of calculating the observabilities from the estimated parameters is described in [16].

The proposed synthesis method begins by applying STAFAN *Stafan()* to estimate the fault detection probabilities and identifying the HTD faults in the given multi-level circuit. Here after, whenever the circuit is changed, there will be a change in the estimated parameters in the circuit. If the testability has to be measured at each iteration of the synthesis process, estimating the parameters every time will be time-consuming. The proposed synthesis method uses *Update_Stafan()*[7], where the earlier estimated parameters are incrementally updated based on the change in circuit structure and the detection probabilities are re-evaluated. This estimation of fault detection probability is based on the nature of transformation made on the circuit. As these estimates are approximate, we use simulation for better parameter estimation after a certain number of *incremental updates*.

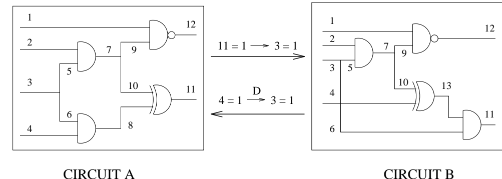


Figure 7: *IB Transformations for Example 3.1*

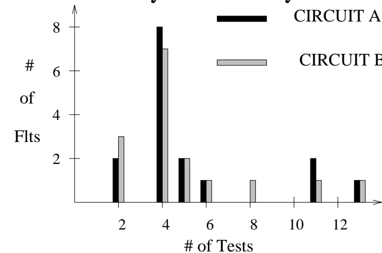


Figure 8: *Detectability Profile*

Testability Cost: The testability cost (TC) takes into account the change in the fault detection probabilities of the HTD faults, as these faults impact on the random pattern test length. In the proposed method, HTD faults are those faults whose fault detection probailities lie between PD_min and $(PD_min \times PD_range)$. PD_min is the PD of the node with the minimum fault detection probability. PD_range is a predetermined value, and depends on the nature of the circuit. Thus, the TC of a transformation is given by :

$$TC = \sum_{\forall l \in HTD} \text{NEW_PD(l) - OLD_PD(l)} \qquad (2)$$

6.2 Testability Enhancement Algorithm

Proposed is an algorithm that can be used to optimize area with enhancement of random pattern testability. IB transformations which enhance random testability can be classified into two types:
(1) Type A: IB transformation enhancing testability with no increase in area.
(2) Type B: IB transformation enhancing testability with increase in area.
The proposed method uses both these transformations iteratively in the synthesis process. It has been observed that as a circuit is transformed during logic synthesis to improve area or speed, its testing property can change, often in unpredictable ways [15]. The most important factor in such a procedure is the ability to identify transformations that are good for the random testability of the circuit.

Example 3.1: Consider two circuits, A and B, in Fig 7. Circuit A can be transformed to Circuit B using the transformation based on (line 11 = 1) → (line 3 = 1). Furthermore, circuit B can be transformed back to circuit A using the transformation (line 4 = 1) $\xrightarrow{D}$ (line 3 = 1). It can be observed that both the circuits have the same area in terms of *connections* and *literal counts*. Here, one can verify whether one of the circuits is better than the other in random pattern testability. The *detectability profile* of the circuits is presented in Fig. 8. Note that circuit A has two HTD faults (faults with only two test vectors) compared to circuit B having three HTD faults. If we assume that

```
     LOT - Random Testability(Iter)
     Begin
1.   Apply Stafan() on the initial circuit;
2.   Phase_Status = Phase 1;
3.   For i = 1 to Iter do
4.     If (Phase_Status = Phase 1) Then
5.       For each line y in the circuit do
6.         Make Transformations 1 - 8;
7.     Else
8.       For each line y in the circuit do
9.         Make Type B Transformations;
10.      For each line y in the circuit do
11.        Make Type A Transformations;
12.    If (Testability reduced in last loop) then
13.      Phase_Status = Phase 2;
     End
```

Figure 9: Algorithm LOT for Enhancing Testability.

all the test vectors are equiprobable, circuit A is a more randomly testable circuit compared to circuit B. Any synthesis method optimizing area or delay may choose circuit B instead of circuit A and degrade random pattern testability of the circuit. If the circuit is a sub-circuit of a larger circuit, the test vectors at the input of the sub-circuit may not be equiprobable and the random testability might be different.□

IB transformations of *Type A* are the area-optimization Transformations 1 - 8. In the proposed method, among these, only those transformations are accepted where the estimated *testability cost* (equation 2) is positive. Thus, area optimization is allowed with improvement of random pattern testability of the circuit.

IB transformations of *Type B*, where the area can be increased to enhance tesatbility involve addition of redundant lines which improve the *observability* of areas in the circuit containing a high concentration of HTD faults. Addition of redundant lines will be based on *Transformations 1 - 4*. Figure 6 gives the example of *Transformation 1*. Note that such a transformation enhances the observability at the vicinity of the fanin cone of f : the shaded area in the Figure. If this area consists of considerably fewer HTD faults, the transformation does not make any improvement in the random testability of the circuit. Transformations where the area at the vicinity of the cone of f has a large concentration of HTD faults and where y has a high observability, are useful in enhancing random pattern testability. The procedure tries to introduce a large number of such redundant lines and ensures that these lines are not removed in the next few iterations during redundancy elimination.

The synthesis procedure, as shown in Figure 9, has two Phases. In Phase 1, area optimization is done on the circuit based on the earlier proposed IB transformations (*1 - 8*). In Phase 2, we use Type B and Type A transformations in order to realize testability enhancement. The synthesis procedure starts with Phase 1 and continues reducing the circuit size until further reduction would reduce its testability. At this point, the procedure changes to Phase 2 (steps 12-13) and continues with transformations with testability enhancement (steps 8-11). Phase 1 minimizes the area without degrading the random testability whereas Phase 2 tries to improve the random testability.

Though the above procedure concentrates only on the testability, the emphasis on area optimization can also be easily controlled. Our synthesis method can iteratively use Phases 1 and 2 until the design goals are met. Moreover, in this synthesis methodology we have not considered the delay of the resulting circuit. The IB transformations presented here can also be used under the added contraint of the delay of the network.

Experimental Results

The PLA benchmarks were chosen for random pattern testability enhancement because these circuits and their multi-level implementation by SIS [10] have poor random-pattern testability [14]. The synthesis tool, LOT, takes in as its input the PLA benchmarks synthesized by *tstfx* [14] and optimizes the resultant multi-level circuit enhancing random testability.

To make the circuits compatable to our implementation, the multi-level circuits synthesized by *tstfx* were mapped to a SIS library obtained from mcnc.genlib; they contain no complex gates. The mapped circuits were then synthesized by LOT. For the sake of comparison, these circuits were also synthesized by HANNIBAL [3], which only optimizes area. The random testability of all the circuits was obtained by using LFSR-generated pseudo-random sequences. The test sequences for *100% single stuck-at fault coverage* presented are an average, over several different LFSR-generated sequences (typically, 5-10 primitive polynomials were used). The circuit areas (Ar) were compared based on 2-input GEs, as was done in the earlier section. The results are presented in Table 4. At the time of writing of this paper, we did not have complete results on two well-known random pattern resistant ISCAS 85 circuits (c2670 and c7552). Preliminary results indicate significant reduction in the fraction of undetectable faults, when compared against original circuits.

Comparison with *tstfx* : As can be observed from Table 4, the proposed method improves both the area and the random pattern test length of the circuits synthesized by *tstfx*. The area of the *tstfx*-synthesized circuits was further reduced by 36.1%, on the average, by LOT. For 7 out of the 10 circuits, there was a considerable reduction in the test length by the synthesis tool - the average ratio being equal to 4.32:1. In particular, circuits with a high test length (in4, in7, vg2, x1dn) have shown considerable improvement in random testability.

Comparison with *HANNIBAL*: HANNIBAL optimizes circuits only on the basis of area. When the same circuits are optimized for random testability, there is a slight increase in the area of the circuits, as shown in Table 4. However, the increase in area is not significant - with the average being equal to 8%. On the contrary, the test length for 100% fault coverage was reduced by 37%. Though in many circuits, optimization by HANNIBAL can reduce the test length, the test length was found to increase in 3 out of the

Circuit	tstfx		Hannibal	Name	LOT		tstfx/LOT		Hannibal/LOT	
	Ar	TL	Ar	TL	Ar	TL	Ar	TL	Ar	TL
b10	382	18.2K	280	18.0K	302	13.9K	1.22	1.31	0.93	1.30
b4	250	47.5K	179	57.0K	189	47.5K	1.33	1.00	0.95	1.20
gary	424	16.8K	288	17.6K	345	15.8K	1.23	1.00	0.89	1.12
in2	368	14.8K	180	12.9K	187	5.4K	2.04	2.76	0.95	2.39
in4	440	490K	197	38.5K	193	38.5K	2.27	12.7	1.02	1.00
in5	227	44.2K	159	34.4K	164	10.1K	1.38	4.36	0.97	3.39
in6	256	47.5K	180	57.0K	192	47.5K	1.20	1.00	0.94	1.20
in7	100	141K	59	53.4K	69	32.8K	1.26	4.29	0.86	1.62
vg2	177	857K	61	95.7K	61	95.7K	2.70	8.95	1.00	1.00
x1dn	251	707K	69	229K	93	143K	2:70	4.95	0.75	1.60
Average							1.56	4.32	0.93	1.58

Table 4: 100% Fault Coverage Test Length and Area Comparison.

10 circuits.

Comparison with *SIS*: It may be noted that circuits synthesized by LOT are much more randomly testable and area-comparable to those synthesized by SIS.

Therefore, in summary, LOT achieves significantly better testability with area, comparable to HANNIBAL. It also achieves significant reduction in area and testability, compared to *tstfx*, which is based on SIS.

7 Conclusions

A novel optimization procedure using recursive learning-based implications is presented here. The transformations proposed are based on (a) new logic transformations based on EX-OR gates. (b) methods to guide the synthesis process enhancing random pattern testability and (c) heuristics to identify difficult-to-test faults. The new EX-OR logic transformations developed with the primary objective of enhancing observability difficulties also play an important role in area optimization. Our synthesis scheme can be applied to poor random-testable multi-level circuits which were synthesized by another synthesis tool to improve random testability, as well as area. Experiments indicate improvement over other optimization tools in terms of area as well as testability.

References

[1] W. Kunz, D.K. Pradhan, "Recursive Learning: A New Implication Technique for Efficient Solution to CAD Problems - Test, Verification and Optimization," IEEE Trans. on CAD, Sept., 1994.

[2] D.K. Pradhan, W.Kunz, "Method for Circuit Verification and Multi-level Circuit Optimization based on Structural Implications,", Patent Application # 08/263,721, Filed date June, 1994.

[3] W. Kunz, P.R. Menon, "Multi Level Logic Optimization by Implication Analysis," ICCAD 1994.

[4] K.T. Cheng, L.A. Entrena, "Multi-level Logic Optimization by Redundancy Addition and Removal," EDAC, pp. 373-377, 1993.

[5] S.C. Chang, M. Marek-Sadowska, "Perturb and Simplify: Multi-Level Boolean Network Optimizer," ICCAD, pp. 2-5, 1994.

[6] S. Muroga *et. al*, "The Transduction Method - Design of Logic Networks Based on Permissible Functions," IEEE Trans Compt., Oct 1989.

[7] M.Chatterjee, D.K.Pradhan, W.Kunz, "ATPG -based Transformations for Random-Pattern Testable Logic Synthesis," Tech. Report 95-024, Computer Sc., Texas A&M University, 1995.

[8] J. Rajski *et. al*, "Testability Preserving Transformations in Multi-Level Logic Synthesis," Intl. Test Conference, pp. 265-273, 1990.

[9] B. Rohfleisch, F. Berglez, "Introduction of Permissible Bridges with Application to Logic Optimization after Technology Mapping," Proc. EDAC/ETC/EUROASIC, 1994.

[10] R.K. Brayton, *et. al*, "MIS: Multi-level Interactive Logic Optimization System," IEEE Trans on CAD, pp. 1062-1081, Nov., 1987.

[11] G. De. Micheli, "Synthesis and Optimization of Digital Circuits," McGraw-Hill, Inc. 1994.

[12] A. Krasniewski, "Can Redundancy Enhance Testability ?," Intl. Test Conference, 1991.

[13] N. Touba, E.J. McCluskey, "Automated Logic Synthesis of Random Pattern Testable Circuits," Intl. Test Conference, pp. 174-183, 1994.

[14] C.H. Chiang, S.K. Gupta, "Random Pattern Testable Logic Synthesis," ICCAD 1994.

[15] M.J. Batek, J.P. Hayes, "Test-Set Preserving Logic Transformations," DAC, 1992.

[16] S.K. Jain, V. D. Agrawal, "Statistical Fault Analysis," IEEE Design Test Comp., pp. 38-44, **2**(1).

[17] S. Gupta, "Synthesis of Testable Combinational Circuits: An Overview of University Activities," Test Synth. Seminar, Intl. Test Conference, 1994.

[18] T. Sasao (ed.) *Logic Synthesis and Optimization*, Kluwer Academic Publishers, 1993.

[19] T. Sasao, P. Besslich, "On the complexity of MOD-2 sum PLA's,"IEEE Trans on Comput., vol. 39, No. 2, Feb. 1990.

[20] D.K. Pradhan, "Universal Test Sets for Multiple Fault Detection in AND-EXOR Arrays", IEEE Trans. Compt., Vol. C–27, pp. 181–187, Feb 1978.

[21] M. Abramovichi, M.A. Breuer, A.D. Friedman, "Digital Testing and Testable Design," Compter Science Press.

[22] A. Sarabi, et. al, "Design of Testability Properties of AND/XOR Networks," IFIP WG 10.5 Workshop on Applications on Reed-Muller Expansion in Circuit Design, Germany, Sept., 1993.

[23] J. Hartmann, G. Kemnitz, "How to do weighted random testing for BIST ?," ICCAD, 1993.