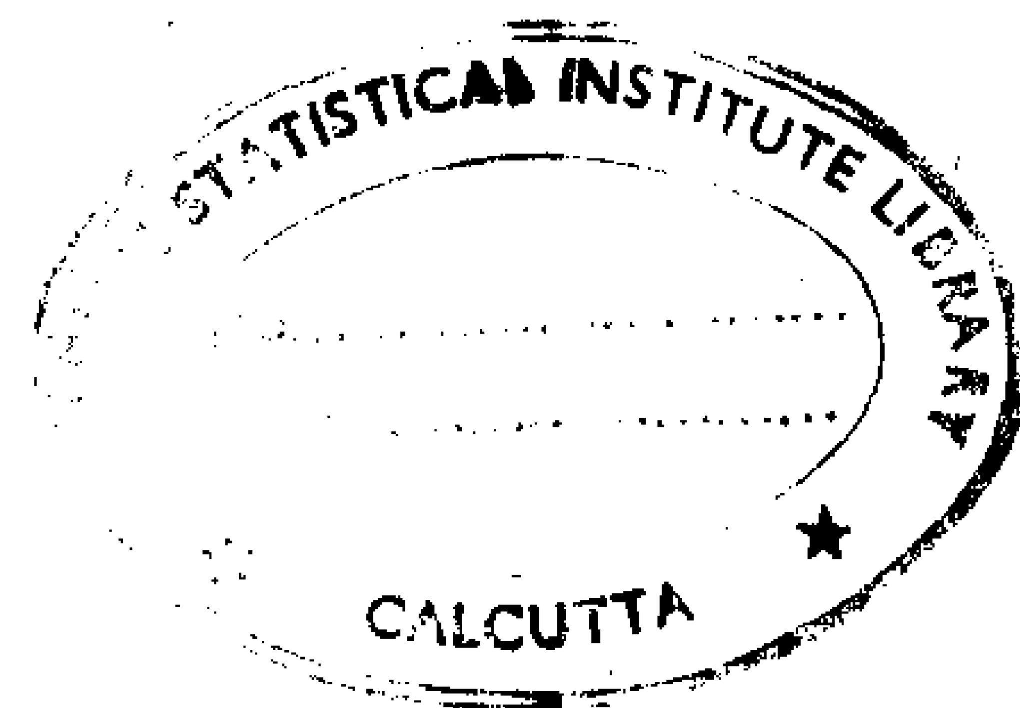
EFFECT OF CIRCUIT STRUCTURE ON PATH DELAY FAULT TESTABILITY IN VLSI DESIGN

a dissertation submitted in partial fulfilment of the requirements for the M.Tech. (Computer Science) degree of the Indian Statistical Institute

By

Biplab Sarkar (MTC9619)

Prof. Bhargab B. Bhattacharya
Advanced Computing and Microelectronics Unit.
INDIAN STATISTICAL INSTITUTE
203 , Barrackpore Trunk Road
Calcutta - 700035



1998

Certificate of Approval

This is to certify that the project work entitled *Effect of circuit structure on path delay fault testability in VLSI design* submitted by Biplab Sarkar , in partial fulfilment of the requirements for M .Tech in Computer Science degree of the Indian Statistical Institute , Calcutta , is an acceptable work for the award of the degree.

Date : July 29 , 1998.

(Supervisor) (External Examiner)

Acknowledgement

I pay my sincerest gratitude to Prof .Bhargab B. Bhattacharya , for his guidance , advice , enthusiasm and support throughout the course of this dissertation .

I would also like to thank Dr. Subhas Chandra Nandy , one of my teachers for his valuable support during the course of this project .

My special thanks to the CSSC unit for providing me with computing facilities.

I thank all of my classmates , who gave me numerous suggestions during my project , and also for a friendly atmosphere during my two years at ISI , Calcutta .

Finally , I express my heartiest thanks to the members of the M . Tech . Dissertation Committee.

INDEX

	Page
1.1 Introduction	3
1.2 Related Works	4
2.0 Definitions and Notations	5
2.1 General Transformations	9
3.0 Algorithm	10
4.0 Experimental Results	16
5.0 Conclusion	20
6.0 Reference	21
7.0 Software	21

Abstract

Failure that cause logic circuits to malfunction at the desired clock rate and thus violate timing specifications are currently receiving much attention .Such failures are modeled as delay faults. They facilitate delay testing. Since design for testability is the approach followed now-a-days several synthesis for path delay fault (PDF) testability is studied in much depth. Local transformation is one such approach. A new approach for applying local transformations is considered here. This approach can be used along with the existing local transformation approaches to get better results.

Additionally , a software implementation of the proposed algorithm is also considered here and the experimental results are quiet attractive in comparison to existing local transformation algorithms.

Keywords: *design for testability , path delay fault model , testability preserving transformation, testability improving transformations .*

1.1 Introduction

Testability is one of the most important aspects during the design and production of Integrated Circuits. All circuits are to be tested for its correctness of behaviour after designing is over. But in order to keep testing cost to the minimum designers have to consider testability aspect very early in the design phase itself . For this fault models are defined and tests for faults in the fault models are conducted. A large amount of physical defects are not actually covered by static fault models, which only checks the logical behaviour of the circuit to be tested. Digital system designers always tries to maximise the system clock frequency in order to extract the maximum performance from the hardware. The maximum allowable clock rate is determined by the propagation delays of the combinational logic part of the circuit . The propagation delay must be lower than the system clock interval in order to avoid delay faults affecting the circuit performance.

Keeping all these aspects in mind more powerful fault models like the Path Delay Fault (PDF) [Smith85] is used , which covers a lot of other fault models e.g. the stuck-at, the stuck-open and the stuck-on fault model.

1.2 Related Works

[Becker95] applied Local Transformations to enhance path delay fault testability. A System for Applying Local Transformations (SALT) was also introduced . A number of local transformations were proposed viz. T_1 , T_2 , T_3 , T_4 & T_5 . (Table 1) .[Karkare97] also proposed some more new transforms. The most unfortunate part of this method is that one has to maintain a database of the possible transformations and try to apply those transformations by checking the existing circuit. This method doesn't seem to be very impressive . But the performance of SALT is impressive for the test circuits that it has been tested upon. The algorithm to be presented in this thesis tries to look at this from a different perspective. The heuristic is to avoid redundancy of logic in the circuit. It is also discussed how to incorporate the existing SALT database technique to this new algorithm thus giving a more powerful algorithm.

Table 1. *Compilation of specific transformations from [Becker95].*

T_1	$(ab)A_1 + (ab)A_2 + \dots + (ab)A_n$	$CA_1 + CA_2 + \dots + CA_n$ with $C = ab$
T_2	$A_1 B_1 + A_1 B_2$	$A_1 (B_1 + B_2)$
T_3	$abA + a'B + b'B$	$MA + M'B$ with $M = ab$
T_4	$(ab' + a'b)A + (a'b' + ab)B$	$MA + M'B$ with $M = ab' + a'b$
T_5	$a'bA + acA + a'b'B + ac'B$	$MA + M'B$ with $M = a'b + ac$

Before proceeding further some definitions and notations are presented here . These terms and notions are used throughout the text.

2.0 Definitions & Notations

A Combinational circuit is modeled as a directed acyclic graph whose nodes are labelled with a basic cell type (AND , OR, NAND , NOR , XOR, XNOR, NOT) or with the name of a primary input or a primary output .

It is well-known that, even if the circuits are correctly designed , a fraction of them will behave faulty because of physical defects caused by imperfections during the manufacturing process. Fault models which covers a wide range of possible defects have to be defined and tests for faults in the fault models have to be constructed . The fault model adopted here is the Path Delay Fault model. Unfortunately , fault coverage for Path delay Fault model is quiet poor for typical combinational logic circuit [Reddy87][Fuchs91] .Therefore , several synthesis approach for PDF Testability have been developed . most of these approaches use Testability Preserving Transforms (TPT) and Testability Improving Transforms (TIT) .

- Path Delay Fault

In Path Delay Fault model , any path with a total delay exceeding the system clock interval is said to have a path delay fault. This models distributed defects that affect an entire path . For each physical path , two corresponding delay paths , the

rising path and the falling path is defined. The rising (falling) path is the path traversed by a rising (falling) transition at the input of the path and changes the direction of transition whenever it passes through an inverting gate .

- Size of a circuit

The Size of a combinational circuit is the sum of the numbers of inputs for all cells in the circuit. (note : Buffer is not taken as a cell) .

- Path in a circuit

A path P of a combinational logic circuit is given by an alternating sequence of nodes and edges ($v_0, e_0, v_1, e_1, \ldots, v_n, e_n, v_{n+1}$) starting at a primary input v0 and ending at a primary output v_{n+1} . Inputs of nodes on the path where no edges e_i the path ends are called *side inputs* .

A rising ($0 \rightarrow 1$) or falling ($1 \rightarrow 0$) transition at v_0 (primary input) propagate along P if a sequence of transitions $t_0, t_1, \ldots, t_{n+1}$ occur at the nodes $v_0, v_1, \ldots, v_n$, v_{n+1} such that t_i occurs as a result of t_{i-1} .

For the detection of a PDF a vector pair $< t_1, t_2 >$ is required rather than a single input vector as in static model. t_1 (t_2) is called the initialization (propagation) vector . In the following some further classifications of PDF are given.

- Robust PDF Test

A vector pair $< t_1, t_2 >$ is called a *robust* PDF test, if the considered PDF is detected by $< t_1, t_2 >$ independent of all other delays in the circuit and all other delay faults not located on P.

- Hazard-free Test

A PDF test for a path P is said to be *hazard-free* if no hazards can occur on P during application of the test.

- Single (Multiple) Path Propagation

A PDF test is called single (multiple) path propagating if a transition is propagated to a primary output through exactly (more than) one path.

- Single (multiple) Input change

A vector pair $< t_1, t_2 >$ is called a *single (multiple) input change* test , if t_1 and t_2 differ at exactly (more than) one position.

- Controlling Value

A *controlling value* at the input of a node is the value which completely determines the value at the output. e.g. 1 (0) is the controlling value for OR (AND) and 0 (1) is the *non-controlling* value for OR (AND).

- Static Value

The value of a signal is called *static* if it remains stable during the application of the initialization and the propagation vector, i.e. the value is not invalidated by *hazards or races.*

- ON (OFF) Implicant

An input vector $t \varepsilon \mathbf{B}^n$ of function is called ON (OFF) *Implicant* , iff f(t) = 1(0).

Using a single path propagating PDF test for one transition along a path P every side input has a static non-controlling value. A single propagating PDF test for the complementary transition along p can simply be obtained by interchanging the initialization and propagation vector .

- Testable (Untestable)

A transition propagating along p is called *testable* iff a PDF test for transition exists ; otherwise the transition is called *untestable*.

According to the above mentioned correspondence a path P is testable , iff both transitions propagating along P are testable .

- Complete Test Set

A set T of vector pairs is called a *complete test set* if it contains a PDF test for every testable path in the circuit . If there is a test for every path, T is called a *full test set* .

- Local Transformations

A *Local Transformation* is the substitution of subcircuits by new realizations which have better properties with respect to the current goal of optimization. The *search pattern circuit* (SPC) represents the subcircuit to be transformed and the *substitution pattern circuit* (SubPC) describes the subcircuit which will be substituted for the SPC .

Now we look at the conditions to be checked to ascertain that a transformation *preserves* PDF testability .

Theorem 1. *Let T be a full test set for a circuit N_1 and let K_1 be a single output subcircuit of N_1 . If conditions* (i) , (ii) *and* (iii) *hold for a circuit K_2 then T is a full test set for the circuit N_2, which originates from N1 by substituting K_2 for K_1 .*

(i) *K_2 has the same input sequence and performs the same function as K_1.*

(ii) *If a test set T_1 tests all paths in K_1 then T_1 tests all paths in K_2.*

(iii) *If a vector pair $< t_1, t_2 >$ produces a static value at the output of K_1 then $<t_1,t_2>$ produces a static value at the output of K_2.*

Proof. Details of the proof is in [Becker95]. Q.E.D

A local transformation is said to *improve* PDF testability if condition (ii) in Theorem 1 can be strengthened to

(ii) *If a test set T_2 tests all paths in K_1 then a proper subset T_2 T_1 exists such that tests all paths in K_2 .*

- Testability Improving Factor

Let T be a local transformation of the SPC K_1 by SubPC K_2 and for all inputs A of K_1 and K_2 let $P_A(K_i)$ be the number of paths in starting at input A and ending at the output of K_i. Then the *testability improving factor* of input A is :

$$\text{TIF}_T(A) = P_A(K_1) / P_A(K_2)$$

2.1 General Transformations

De Morgan's laws and factorization are the basic transformations used in the synthesis process. Its is well-known that de Morgan's laws and factorization retain PDF testability [Devadas90]. A number of transformations are given in [Becker95] and [Karkare97] . It was shown in [Becker95] that distributive law may not preserve PDF Testability. It may introduce some hazards to the transformed circuit . The proposed algorithm uses these basic transformations viz. De Morgan's law and factorization for minimization.

3.0 Algorithm

The basic heuristics in the proposed algorithm for PDFT Enhancement (PDFT_En) is to see that the circuit minimizes the number of redundant logic. Since in general , the redundancy in a circuit is leading to untestability of a circuit path . So ,one can have a non-redundant circuit with all the paths sensitizable which will help in making PDF testability possible .This idea has made the algorithm to go for an incremental build-up of the circuit starting form the primary inputs and gradually adding more and more logic gates to it taking care that redundant logic is avoided at every stage of insertion of new logic gate. This means *create a new logic gate in the circuit if that logic doesn't exist in the circuit constructed so far* . The algorithm assumes that a logic may exist in one of the following forms in the circuit.

(i) Exactly in the same form as one is trying to look for or as its inverse or de Morgan equivalent or some other equivalent forms.

(ii) Factorise out from some existing logic. e.g. a four input OR is same as a cascading of two or more OR gates with lesser number of inputs.

The main bottle-neck in the implementation of this algorithm is to maintain a fast access database for all the logics created in the circuit so far using some coded notations. Before starting the algorithm some preprocessing is to be done on the input circuit's netlist . The circuit gates are to be levelled such that for any gate G (A, B) having inputs A & B , *level (G) > level (A)* and *level (G) > level (B)* . This levelling is required because the algorithm is an incremental one . It assumes that the logic created so far are made as irredundant as possible , by the logic matching scheme adopted by the algorithm . So, gates are to be inserted in the circuit database according to increasing order of levels , starting form the primary inputs which are given the minimum level value.

The pseudo-code for the algorithm is given here :

Procedure: PDFT_Enhancement

Input: The Circuit Netlist with all gates levelled according to their depth from the input gates.

Output: The Circuit netlist with better PDF Testability .

Aim: The procedure inputs a netlist of logical gates and tries to minimize the netlist inorder to enhance Path Delay Fault Testability by avoiding generation of redundant logics and trying to reuse existing logics .

Variables: *MAXLEVEL* : Gives the maximum number of levels (depth) in the circuit.

MAX_GATE_AT_THIS_LEVEL: Maximum number of gates at a particular level.

G(i , j) : The *i* - th Gate G at level *j* .

start

(Introduce levels in the circuit gates so that the dependencies are easily understood.)

Level_Adjusting( ).

for j = 1 to MAXLEVEL do

for i = 1 to MAX_GATE_AT_THIS_LEVEL do

(See that if the logic generated by this gate exists already in the circuit database constructed so far . Use various equivalent forms of the logic to answer this query. Use some preprocessing of the fanins of the logic so as to minimize the size and the pathcount.Infact use factorization to achieve this.)

Pre_Process(G(i, j)) .

endfor

endfor

(Now the minimised circuit is created . But it exist in the database in a form that makes each gate related to the other only by the fanin relations. The

netlist to be generated must be related by the fanin as well as the fanout from each gate. So a circuit extraction module is called.)

Extract_Circuit( ) .

end

Procedure: Level_Adjusting

Input: The circuit netlist .

Output: The levelled circuit netlist.

Aim: The circuit gates are levelled so that the dependencies are easily understood. Any gate G(A,B) having input A and B will have level(G) > level(A) and level(G) > level(B).

Variables: *G* is a logic gate.

start

Do a breadth-first traversal and set the levels to all gates starting form the primary input side and going towards the primary output.

end

Procedure: Pre_Process

Input: A gate G.

Output: No Output.

Aim: The input logic is tested if it can be minimized using the factorization technique . This infact improves the circuit characteristics. After that, the logic gate is inserted in the netlist by checking if any existing logics can be reused instead of generating a new logic.

Variables: *G* is a logic gate.

start

If the logic G can be factorised

factorise the fanins and INSERT new logics formed in the database .

else create a new logic and INSERT it in the database.

end

Procedure: INSERT

Input: A gate G.

Output: No Output.

Aim: To insert a new logic gate in the circuit database

Variables: G is a logic gate.

start

(Look in the database to see if the logic gate already exists or not.)

if gate G is not found in database create a new entry in the database.

end

Procedure: Extract_Circuit

Input: No Input

Output: Circuit Netlist .

Aim: The circuit formed in the database is inter-related by the fanins only. But

we want to get a netlist having inter-relation by both fanins as well as fanouts . So we start extracting the circuit from the primary output side and gradually come to the primary inputs on the way building the fanouts for each gates .

Variables: G is a logic gate.

start

(Starting from the primary output use breadth-first traversal and form the fanout list forthe gates which are in its fanin list .Gradually build up the fanouts and proceed forth.)

end

The Logic equivalence that is tested here looks into the following equivalent forms:

(i) Exact form or its inverse. e.g. (A + B) is equivalent to (A'. B')' or inverse of (A + B)' or (A' . B') .

(ii) The logic may also exist in association with some other logic. e.g. a 4-input OR gate can be expressed as a cascade of more than one 2-input OR gates.

(iii) There may be some more equivalent forms which may be good. Here the idea of Local transformations may be used . (this is an improvement over the algorithm)

Analysis:

Let us denote the number of gates in the circuit to be n for the next of our discussion. Here we shall be considering the above algorithm considering a simple database . No hashing technique is assumes. A crude way of data accession is considered .

Lemma 1. *The Size of the database is of order $O(n^2)$.*

Proof. *Since at each iteration we want to either reuse some existing logic or try to build some new logic, there is at maximum insertion of **n** new gates in the database at each iteration. This insertion takes place when factorization is done on the fanins . As the fanins are of the order $O(n)$.* Q.E.D.

Lemma 2*. The complexity of the search (INSERT) algorithm is of the order* $O(n^3)$.

Proof. *Since the size of the database which contains the gates is of the order $O(n^2)$ the complexity for any search is the complexity for finding out the gate and the complexity to compare the fanins. Each gate can have fanins of the order $O(n)$. So the complexity is of the order $O(n^3)$ considering linear access time for finding out the gate of interest and looking into the fanin of the gate..*

Lemma 3. *The complexity of the Level_Adjusting and Extract_Circuit is of the order* $O(n^2)$.

Proof. *Since a breadth-first search is only used.* Q.E.D

Theorem 2. *The complexity of the PDFT_Enhancement algorithm is of the order* $O(n^4)$.

Proof. *The algorithms inner loop is having a complexity of the order* $O(n^3)$ *. The outer loops are in total for **n** times .*

Q.E.D

If hashing technique is used the accession time from the database can be reduced to a constant time. This would have resulted in order $O(n)$ looping followed by order $O(n^2)$ breadth-first searches giving the overall complexity to be of the order $O(n^2)$. The algorithm tries to minimize the use of gates . The use of NOT gates are minimizes as and when possible. BUFFER gates may be used if requires as it is assumes that primary outputs don't have any fanouts. In all cases the size and the number of paths are minimised but never increased. The depth of the circuit may be changed . The number of primary outputs may be reduced . In such case more than one output gets merged into one output . The number of primary may also get reduced as a bidirectional check is used to see if all outputs and inputs are infact useful or not. If an input is unable to cause any effect on the output side it is removed. Similarly the outputs are also checked .

Improvement:

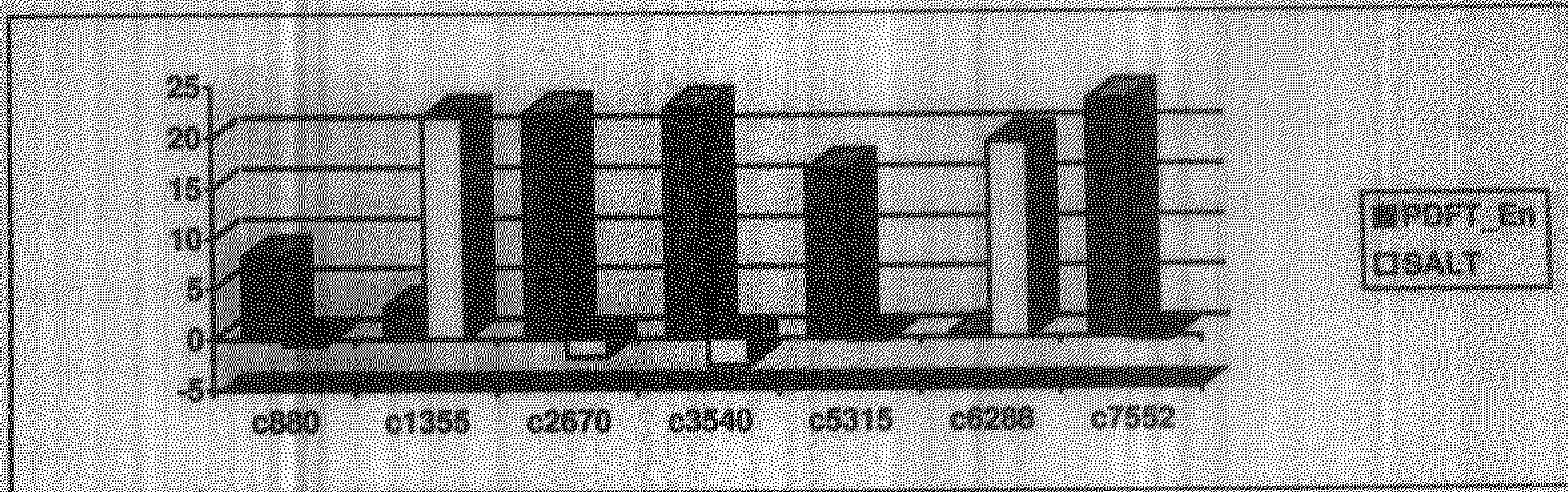
The Algorithm is robust enough to be able to incorporate the local transformation algorithm of PDFT enhancement as proposed by [Becker95] but using somewhat a bigger search space for the substitution circuit. In the implementation two transformations viz. T_4 & T_5 as proposed by [Becker95] are introduced as a case study . By default the transformations T_1 , T_2 & T_3 are taken care of be the algorithm itself without any special treatment. So one can improve upon this algorithm by using a good data accessing technique and incorporation the local transformation technique as proposed by [Becker95] to get a better result.

4.0 Experimental Results

The experimental results are quiet impressive when compared with the results obtained by [Becker95] using SALT. So a comparative tabulation (Table 2) is done here. The ISCAS85 benchmark circuits are used for testing. The implementation was done in C++ on a Sun SPARCStation-1 under SunOS-4.0.x. having 4MB RAM . Because of the hardware constraints the CPU timings may not be accurate (enough swapping.). The timings for the main algorithm is only given , the post processing times are not included here.

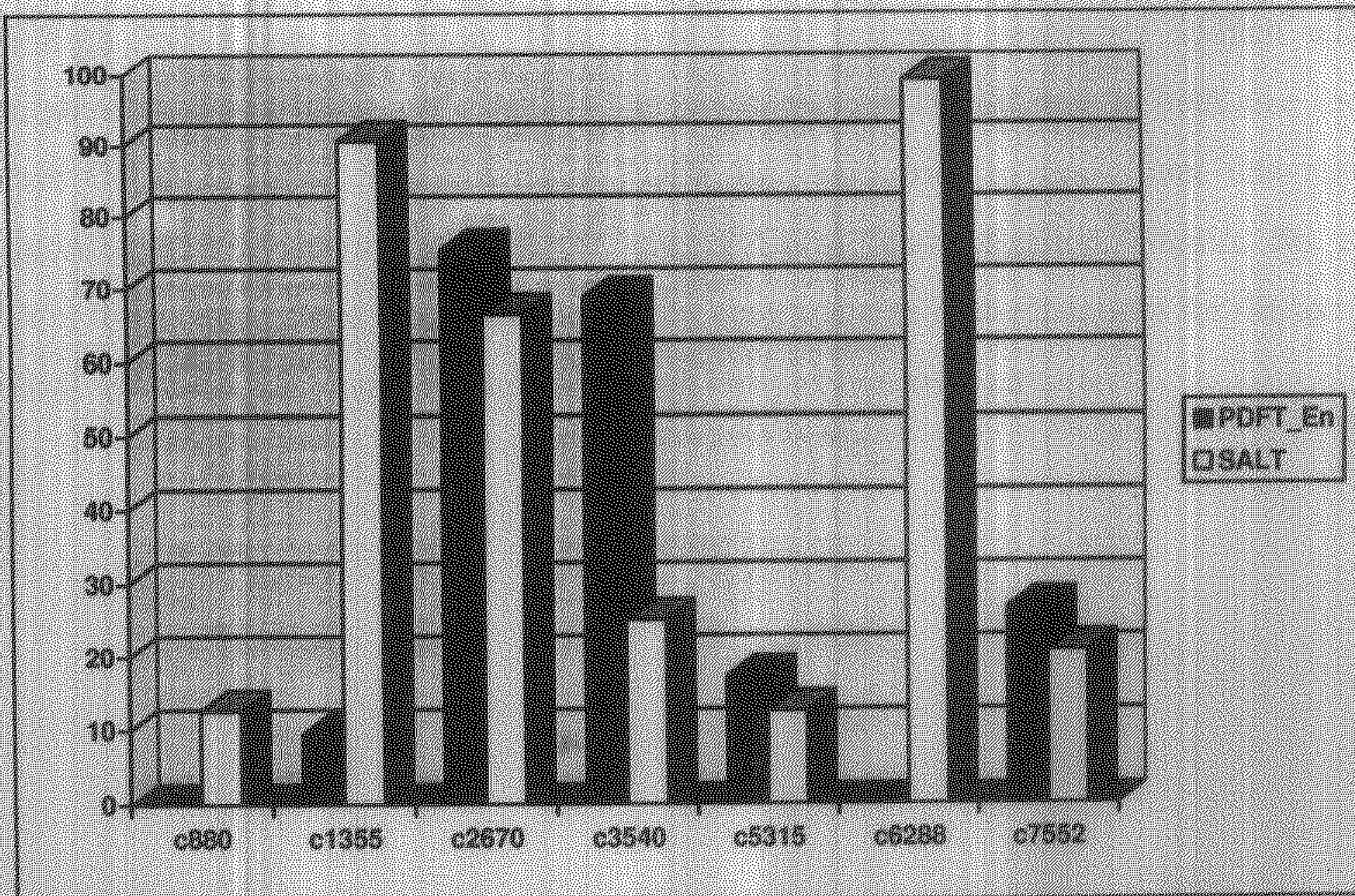
Table 2. Results of Transformation on different ISCAS85 circuits.

Circuit	Algorithm	Size	Path	# [illegible]	Depth	CPU sec
c17	Original	13	11	6	4	0.022
	SALT	---	---	---	---	---
	PDFT_En	12	11	6	4	0.049
c880	Original	703	8642	383	25	1.125
	SALT	707	7546	---	29	---
	PDFT_En	645	8642	333	22	2.246
c432	Original	336	83926	160	18	0.253
	SALT	---	---	---	---	---
	PDFT_En	309	83926	136	15	0.539
c499	Original	376	9440	170	10	0.359
	SALT	---	---	---	---	---
	PDFT_En	344	8544	142	12	0.496
c1355	Original	1032	4173216	546	25	2.436
	SALT	804	397888	---	19	---
	PDFT_En	1000	3775776	518	27	5.482
c1908	Original	1330	729056	875	41	7.535
	SALT	---	---	---	---	---
	PDFT_En	955	726965	597	42	6.014
c2670	Original	1829	679954	1143	32	18.184
	SALT	1862	228727	---	32	---
	PDFT_En	1420	168252	919	36	30.231
c3540	Original	2655	28265874	1611	48	35.435
	SALT	2726	21140286	---	52	---
	PDFT_En	2050	8845346	1163	52	40.694
c5315	Original	4062	1341305	2296	50	77.569
	SALT	4069	1173865	---	49	---
	PDFT_En	3385	1111569	1960	52	80.004
c6288	Original	4800	9×10^{19}	2416	125	100.050
	SALT	3872	8.70×10^{15}	---	94	---
	PDFT_En	4784	9×10^{19}	2400	125	179.469
c7552	Original	5573	726494	3475	44	185.711
	SALT	5588	576694	---	44	---
	PDFT_En	4267	533581	2587	43	87.041



Percentage Reduction of the Size of the circuit.

Figure i



Percentage Reduction in Path Count.

Figure ii

The Size of the circuit is reduced which in most of the cases is better than that done by SALT (Figure i) . The counting of path is done in the post processing phase using simple breadth-first algorithm which is of order $O(n^2)$. The path count is much better in SALT (Figure ii) . Of coures in some case PDFT_En is having an upper hand..

As one can see that the pathcount in PDFT_En is not as good as that of SALT. But the Size is impressive as compared to SALT. This is because PDFT_En is based upon modifying the existing circuit logic only and not adding any thing new. It tries to reuse existing logics. This helps in reducing the size to some extant. But for Path Count reduction PDFT_En has to use factorization as much as it can . So for some circuits the result is not quiet impressive when the Path Count is taken into account. And use of factorization is an inherent property of the circuit and can't be predictable. SALT on the other hand is based upon the philosophy to replace existing circuit module with better modules. So it can focus on some transformations which will result in high Path Count reduction. This gives a new proposed algorithm which will combine both PDFT_En and SALT . The new algorithm should look at both size and Path Count together. The algorithm PDFT_En also tries to implement this new algorithm just by having a small database of Local Transformation Circuits. So, in order to get impressive results one has to increase the size of the database for the local tranfromations. The current implementation has just two such circuits .

5.0 Conclusion

To conclude PDFT_En is an $O(n^2)$ algorithm if implemented carefully. SALT seems to be a good but complex algorithm which can be used on top of PDFT_En as the underlying algorithm thus leading to better results. The hybrid system is having the potential of improving the Path Delay Fault Testability at the same time reducing the Size and PathCount of the circuit.

6.0 Reference

1. [Becker95] ***H. Hengster*** , ***R. Drechsler*** and ***B. Becker*** ." **On Local Transformations and Path delay fault Testability** " . *Journal of Electronic Testing: Theory and Applications,* Vol - 7 ,1995, pp. 173 - 192 .

2. [Smith85] ***G. L. Smith*** , " **Model for Delay Faults Based Upon Paths** ." *Prob. Of Int'l Conf.* 1985, pp. 342 - 349.

3. [Reddy87] ***S. M. Reddy*** , ***C. J. Lin*** and ***S. Patil*** ," **An Automatic Test Pattern Generator for the Detection of Path Delay Faults** ." *Proc. Of Int'l Conf. On CAD* , 1987 , pp. 284 - 287.

4. [Fuchs91] ***K. Fuchs*** , ***F. Fink*** and ***M.H Schulz***, "**DYNAMITE: An Efficient Automatic Test Pattern Generation System for Path Delay Faults.**" *IEEE Trans. On CAD,* Vol - 10 (10) . pp. 1323 - 1335, 1991.

5. [Devadas90]. *S.* ***Devadas*** and ***K. Keutzer***, " **Necessary and Sufficient Condition for robust Delay - fault testability of Logic Circuits.**" *Sixth MIT Conference on Advanced Research on VLSI* , 1990, pp. 221 - 238 .

6. [Karkare97] *A.* ***Karkare*** , ***M. Singha*** and *A.* ***Jain*** . " **Testability Preserving and Enhancing Transformations for Robust Delay fault Testability** ". *IEEE Computers.* 1997. Pp . 370 - 373.

7.0 Software

The source code for the software developed is available free at the author's web site. URL ***http://members.tripod.com/~biplabsarkar/.***