Multi-level Logic Minimization Through Fault Dictionary Analysis

Ronald W. Mehler and M. Ray Mercer

Abstract

This paper presents the results of the study of a new algorithm for multi-level logic minimization.

The study is based upon the premises that an untestable node is a redundant node and that nodes that do not demonstrably cause conflicting behavior at primary outputs may be compatible.

Data gathered using techniques presented here show that fault dictionary analysis is a powerful tool that produces minimization results in selected benchmark circuits superior to any previously published academic work. The algorithm developed in this study, Texas Aggies Logic Optimizing Netlister (TALON), is shown to be competitive with, and complimentary to, other methodologies. TALON can be used by itself to reduce the size of a logic network, or it can be used as a preprocessor or postprocessor for other tools, giving superior results to those obtained by any of them working independently.

Keywords

Multi-level logic minimization, optimization

I. Introduction

Multi-level logic minimization is the key to producing high-quality digital circuits. As designs become ever larger and are produced at higher levels of abstraction, the gate-level implementation must be produced by automated tools. Competitive, time to market pressures preclude hand crafting logic circuits, yet the first-pass translation from hardware description languages to a netlist representation is inefficient in terms of area and low-performing in terms of speed.

Commercial tools currently on the market tend to minimize circuits through manipulation of Boolean expressions, finding common terms that can be factored out. Using binary decision diagrams or similar forms of algabraic manipulation, data structures can grow exponentially with circuit size. Memory limitations bind the space that can be searched for possible reductions.

The algorithm presented here reduces not algebraic terms, but gates in a technology-independent netlist. It does so by finding pairs of gates that produce indistinguishable effects on the primary outputs of a circuit, whether or not the actual internal functions of the gates in question are truly identical.

Potential targets for reduction are identified through their response to test patterns. Single stuck-at fault simulation for a limited test pattern set is used to characterize the full set of stimulus responses. A modified version of the fault simulator TAMUFS[2] is employed to find both good and faulty circuit response. This fault simulator writes out fault dictionaries, or matrices, of both the actual values at each node for each test vector and which stuck-at fault, if any, is detected for each node for each vector.

Both *potential implicant function* pairs and *potentially compatible* pairs are found through analysis of these matrices. Lists of these pairs are then screened to reduce effort spent on unpromising pairs. Promising pairs are checked for correct functioning by employment of a miter circuit.

Logic minimization is an iterative process. After each successful reduction, lists of potential pairs are generated for the new circuit. This process continues until all pairs have been tested.

During preliminary testing of TALON, benchmark circuits were reduced in area by merging compatible gates. The improvements were less than dramatic. However, using stimulus response data to find and insert implied functions to a netlist, then repeating the compatibility search on the resultant netlist, was found to significantly increase the number of gates which could be eliminated. While the percentage of insertions that lead to gate eliminations was small in every case, the large number of implied functions available for testing in the benchmark circuits allowed this technique to produce circuits that eliminated more functions than any other published system.

The rest of this paper is organized as follows:

Section II presents an explanation of test, testability and fault modeling.

Section III shows how the concepts introduced in section two are used by TALON to optimize logic circuits.

Section IV contains experimental results obtained by use of TALON and a comparison of these results to those obtained by other tools.

Section V gives conclusions and suggestions for future work.

II. Test and Testability

When integrated circuits are fabricated, some percentage of them will inevitably be faulty. These manufacturing flaws are most commonly the result of shorts or opens in the metalization. Regardless of the cause of faults, part of the manufacturing process is screening out the defective parts. Although it is certainly in the interest of the manufacturer to analyze failures, simply differentiating between good and faulty parts is a task of significant complexity and one that has received much attention in recent years.

To test for every possible short or open would be an impossibly time-consuming task. Even exhausting the input space for a modestly-sized combinational circuit is daunting: a 32-bit adder with a carry in would take 2^{65} vectors to fully test. At 100 MHz, a tester would labor for over 11,000 years to test a single part. To provide a more reasonable throughput, manufacturers have generally adopted the single stuck-at fault model for testing their production.

There is not an isomorphic relationship between actual faults in manufactured parts and modeled stuck-at faults, but empirical evidence shows that stuck-at fault modeling can be used to detect a satisfactorily high percentage of manufacturing defects. This has led to advances in automatic test pattern generation algorithms, as there is signficant financial advantage to being able to produce small test vector sets, ones that will take minimal time to detect defective parts.

This is illustrated by the transistor-level model of an XOR gate shown in Fig. 1. There is no single stuck-at fault that can accurately model the behavior of this gate with a bridge fault as shown. However, employment of the single stuck-at fault model can probabalistically determine that the gate is not behaving correctly, and, for production testing (acceptance from the fabricator), to date that is sufficient information.

Single stuck-at fault models for an XOR gate are shown in Table I.

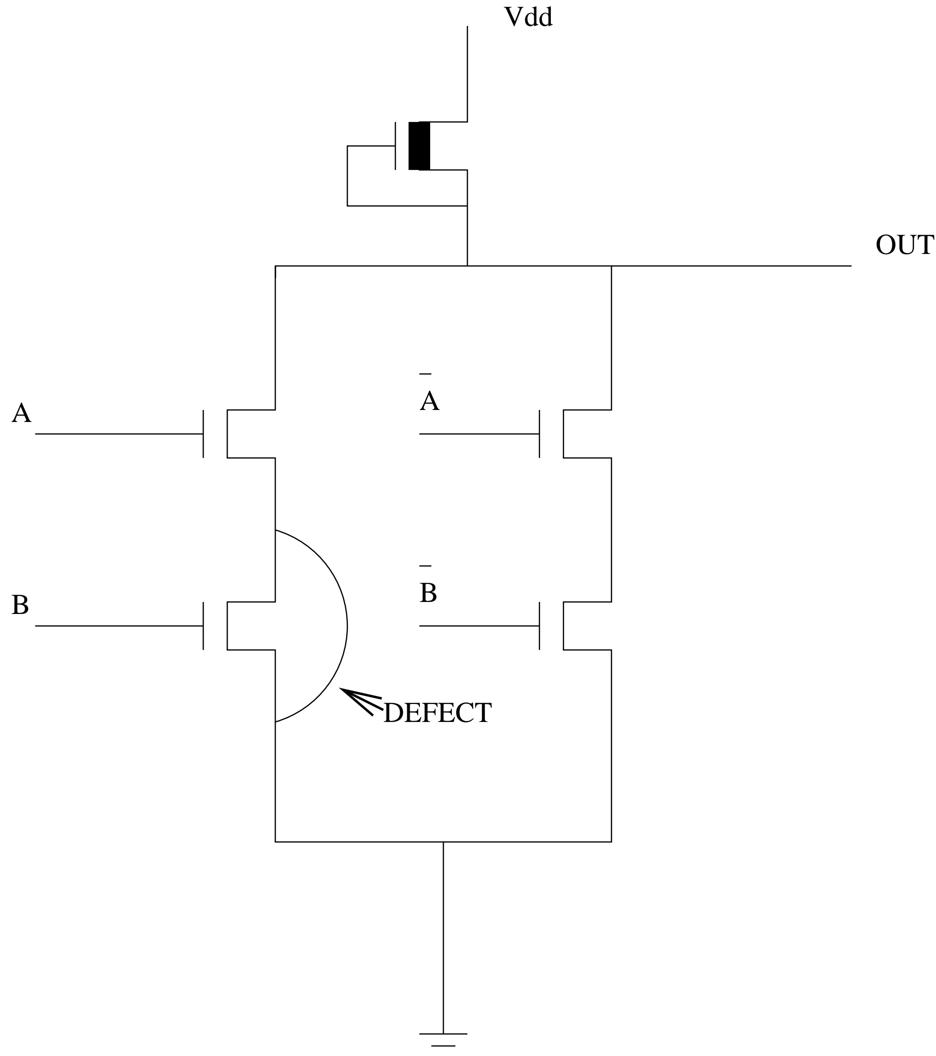

Fig. 1. An XOR gate with a bridge fault

TALON uses the single stuck-at fault model and fault detectability data to find and eliminate redundancies in logic circuits.

A. The Single Stuck-at Fault Model

The single stuck-at fault model compares the outputs of a circuit with a copy of the circuit in which one point is stuck at logic zero or logic one. This procedure is repeated for both one and zero for every node in the circuit. A circuit in which every manifistation of a stuck-at fault produces an output which is distinguishable from a circuit model containing no faults is said to be fully testable.

This is something of a misnomer, as the single stuck-at fault model may fail to uncover some esoteric defects. Nevertheless, this is the fault model that has been adopted throughout the electronics industry.

The 2:1 multiplexor shown in Fig. 2 is fully testable for all single stuck-at faults. Thus holding any input, output or internal node to a fixed value, either logic one or logic zero, will result in the output, for one or more input vectors, differing from the value of a properly-functioning multiplexor. This is shown in Table II.

Fault simulators would typically differentiate between fanout stems and branches. In the case of the multiplexor shown in Fig. 2, this would lead to six possible faults for the select line: stuck at one and stuck at zero for the stem and for each of the two branches.

The 2:1 multiplexor shown in Fig. 3 contains a redundant gate. That this gate prevents a hazard (when the select line switches while the two data inputs are held steady at logic one) does not change the fact that from a static analysis

TABLE I

XOR gate fault models. No single stuck-at fault accurately models the bridge fault shown in Fig. 1, yet complete stuck-at fault modeling would detect that a gate with the indicated fault was not operating correctly. In the above table, A/0 indicates the gate output with input A stuck at logic zero, OUT/1 indicates the output is stuck at logic one, etc. The column labed "FAULT" shows the behavior of the gate with the bridge fault shown in Fig. 1.

B A	Fault-free	A/0	A/1	B/0	B/1	OUT/0	OUT/1	FAULT
0 0	0	0	1	0	1	0	1	0
0 1	1	0	1	1	0	0	1	0
1 0	1	1	0	0	1	0	1	1
1 1	0	1	0	1	0	0	1	0

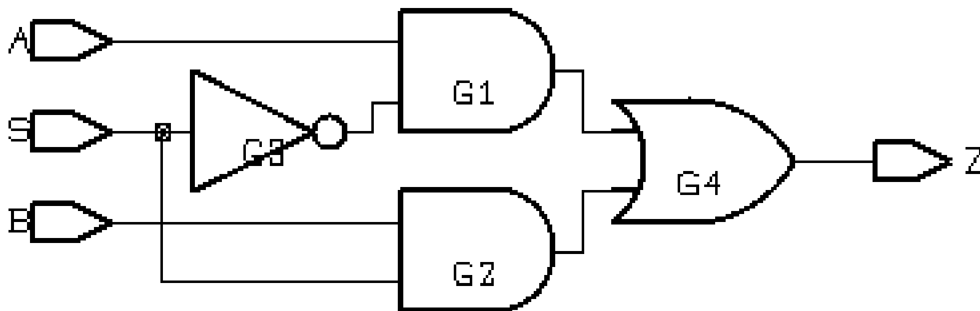

Fig. 2. A fully-testable 2:1 multiplexor

standpoint, the third AND gate is superfluous. No input vector or sequence of input vectors would detect any of several different faults involving this gate.

B. Testablity and Functional Redundancy

In an irredundant logic circuit, each and every input and internal signal can effect one or more outputs.

If, on the other hand, there exists a signal that can be held to any arbitrary value without affecting any output, then that signal is redundant. It is not contributing anything to the output function and can be removed without changing the functionality of the circuit. A logic optimizer should successively find and remove any such signals.

An automatic test pattern generator (ATPG) tool tests the effects of various faults on the outputs of a circuit. If a given node can be faulted (set to a fixed value irrespective of input conditions) without effecting any outputs, then the fault is not detectable, at least not by that ATPG tool. If a node can be set to both stuck at one and stuck at zero without any fault being detected, then there are three possibilities:

1: The node is redundant.

2: The fault simulator is insufficiently powerful to detect the fault.

3: The fault simulator has not worked sufficiently hard to detect the fault, but could be instructed to do so.

Using the PODEM [3] or FAN [3] ATPG algorithms, as is common in today's fault simulators, all detectable stuck-at faults eventually will be detected if backtracking limits are set sufficiently high. If an ATPG tool employing one of these algorithms exhausts the search space without finding any test that will detect any stuck-at fault for a given node, then

TABLE II

Stuck-at fault detection. Single stuck-at faults that would be detected for each input vector.

SBA	A/0	A/1	B/0	B/1	S/0	S/1	G1/0	G1/1	G2/0	G2/1	G3/0	G3/1	Z/0	Z/1
000				X				X		X				X
001		X						X		X				X
010			X			X			X		X		X	
011		X			X			X		X		X		X
100				X		X		X		X				X
101	X				X		X						X	
110			X						X		X		X	
111	X						X						X	

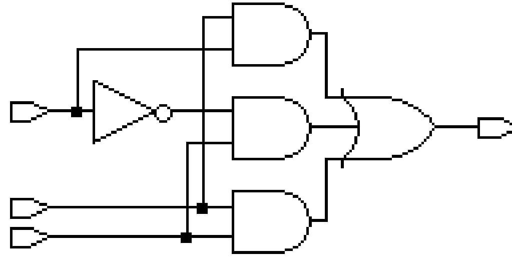

Fig. 3. An untestable multiplexor with a redundant gate

the node is redundant.

C. Compatibility and Fault Detection Analysis

In most examples of logic circuits, totally untestable and thus redundant gates are few and far between. A logic optimizer that only finds and removes such gates will not greatly improve a design.

Compatible gates, gates whose outputs may be substituted for each other without changing the circuit's functionality, are much more common. Compatibility may be bidirectional, in which case either gate may be substituted for the other, or unidirectional. In either case, once a gate has been identified as compatible with another, that gate and all gates that feed only it, may be eliminated, with the output of the target gate replaced by an additional fanout branch from the compatible gate.

Potentially compatible pairs may be found through fault detection analysis. This is the process of examining the fault detection characteristics of each node in comparison to those of other nodes. Applying a series of test vectors, fault detection data for each node are captured. Each data point has three possible values: stuck at zero would be detected, stuck at one would be detected or no stuck at fault would be detected.

Conflicts occur when, for one or more vectors, a pair of nodes would detect different stuck-at faults. Pairs than have no conflicts for an entire vector set are potentially compatible. These pairs are saved and then tested for true compatibility. The test for potential compatibility between two nodes may be aborted upon finding any conflict.

Pairs are determined to be compatible after an exhaustive ATPG tool search fails to uncover any pattern that proves the pair to be incompatible. The search is of a miter circuit specially built for the pair in question. Miter circuits used in TALON are explained in greater depth in the following chapter.

III. The TALON System

TALON consists of a series of programs designed to reduce the number of gates in a netlist, while preserving all of the functionality of the original.

While all metrics gathered in this study refer only to the size of netlists, the nature of the reductions performed would also tend to decrease gate delay across the reduced circuits in comparison to the original netlist.

The phases of a TALON run are as follows:

1: Initial Setup
2: Redundancy and Compatibility Optimization
3: Implicant Search
4: Implicant Testing
5: Redundancy and Compatibility Optimization on Modified Netlist

A. Initial Setup

The first step in the optimization procedure is to form matrices of fault detection data and fault-free binary values.

Using a vector set created from several runs of random pattern generators and fault-targeting ATPG tools, the previously-referenced TAMUFS fault simulator is employed to generate two matrices: one of faults detected and one of the binary values for each node in the circuit under test.

Once the fault detection matrix is transposed for easier processing, a line represents the fault detection status of a node for the vector set. Nodes are then compared to find ones that do not conflict. A redundant node will show up as not detecting any faults. Since such a node will not conflict with any other fault detection profile, it will be eliminated.

A sample matrix for a four gate netlist with five test vectors is shown in Table III.

TABLE III

SAMPLE FAULT MATRIX. A NETWORK OF FOUR GATES IS BEING TESTED WITH FIVE VECTORS.

gate_1	1	0	1	1	1
gate_2	0	0	N	1	1
gate_3	N	0	N	1	1
gate_4	1	N	1	N	1

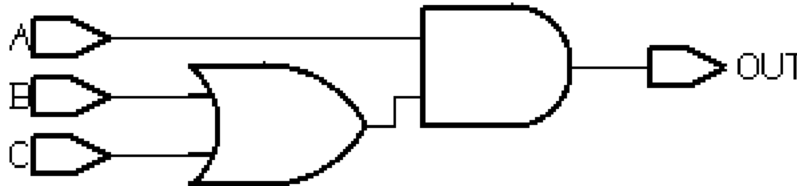

Fig. 4. Potential pair. A pair of potentially-compatible gates.

Entries in this table show the faults that would be detected: a "1" means that stuck-at one would be detected for the associated test vector, "0" means that stuck-at zero would be detected and "N" means that no fault would be detected at that node for that vector.

In this network, gates one and two are incompatible: on the first vector, gate one would detect a stuck at one fault, while gate two would be detecting the stuck at zero fault for the same pattern. It is not necessary to examine this pair any further.

Gates one and three form a "1N" pair: they either have the same fault detection values or gate three would not be visible when gate one detects some fault. They are potentially compatible.

Similar analysis may be performed for all two gate permutations. For this sample matrix, the potential pairs are:

gate_1 gate_3
gate_1 gate_4
gate_2 gate_3
gate_3 gate_4

Although gates one and two are incompatible, both are potentially compatible with gate three. More testing is needed to determine which, if either, is truly compatible with that gate.

Gates three and four have no one-zero conflicts. However, each detects faults when the other does not. The prospects for a successful merge between this type of pair is considered too low to process further and such pairs are not saved for further processing.

Using additional data provided by TAMUFS, positional data from the matrix is correlated to node names in the netlist. Potential pairs of netlist nodes are then written to a file.

Most of these pairs are subsequently screened out before any testing for true compatibility is performed.

A.1 Feedback Screening

Netlist reductions are made by replacing one node with another. If the replaced gate has no other fanout, it may be eliminated from the netlist, along with all gates that serve only to form that signal.

This methodology could form combinational feedback loops. Fig. 4 shows a pair of gates. If dictionary analysis shows they are a potential pair, the replacement procedure would give the feedback loop circuit shown in Fig. 5.

To avoid creating this sort of defective circuit, each merge target is checked for fan in. If the target gate appears in the fan in cone of the replacing gate, the ordered potential pair is dropped. The entire fan in cone, all the way back to primary inputs, must be checked to ensure no combinational feedback loops are created. Dropping one permuation of a potential pair does not preclude further testing of the other permutation.

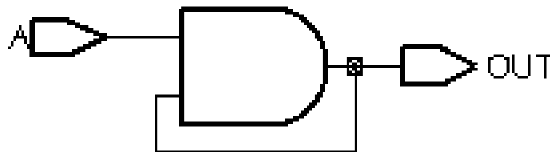

Fig. 5. Feedback pair. Potential pairs that would create feedback loops must be screened out.

A.2 Distance Screening

In [4], "recursive learning" [5] is employed to find logical relationships between nodes, thus identifying good candidates for reduction. TALON takes a simpler approach to finding relationships between potential pairs: topological proximity.

Convergence and divergence distances are calculated for each potential pair. These distances are the number of gates that must be traversed in either direction before reaching a common point. In the 2:1 multiplexor of Fig. 2, the convergence distance of the select line and either data input is two. The divergence distance is infinite, as these primary inputs have no common point of origin.

Empirical data gathered as part of this study show that a potential pair has a vanishingly small probability of being truly compatible if both the convergence and divergence distances are greater than three. These distances refer to the shortest path of the farthest gate to the common point. In the example of the multiplexor, the select line has two paths to the output: one through the inverter and two gates, the other just through two gates. If the select line and the final output were a potential pair, then their convergence distance would be two.

To speed processing, pairs that neither converge nor diverge within three gates are discarded.

A.3 Logical Conflict Screening

It is quite rare to find potential pairs that have identical lines in the fault detection matrix. The vast majority have some entries where one detects a fault and the other does not.

Assume that the OR-AND gates of Fig. 4 form an "N1" pair, that is, the two nodes detect all of the same faults except that for one or more vectors, a stuck-at one fault would be detected at the AND gate, output, while no fault would be detected at the OR.

The gate merging algorithm would suggest substituting the output of the AND gate for the output of the OR gate. Since this would create a feedback loop, this pair will be discarded during feedback screening.

However, it may also be possible to substitude the OR gate for the AND. By removing the AND gate, the observability of the OR changes. Unless a merge involves a primary input, all potentially compatible pairs are initially assumed to be bidirectional. Any ordered pair that would require substituting a gate output or another primary input for a primary input is immediately discarded.

Logical screening may be performed on the OR-AND pair to determine if it is promising.

In addition to a fault detection matrix, the TAMUFS fault simulator also writes out a matrix of the binary values of each node for each test vector. For vectors where "N1" potential compatibility is found, the actual binary values of each node are compared. If they are identical, then changing the observability of the non-detected node by removing the detected one has a high probability of leading to a valid area reduction. Conversely, if the binary values conflict in those spots, the pair is investigated no farther due to the miniscule probability of a successful reduction.

A.4 Effort Reduction Through Screening

A modestly-sized combinational circuit such as the ISCAS benchmark circuit c432 [6] (160 gates, including single-input gates) will produce hundreds of potential pairs.

Employing these screening methods reduces the number that will need to be checked by more time-consuming methods by at least a factor of three. In some cases, the hundreds of pairs originally generated are reduced to single digits.

B. Redundancy and Compatibility Optimization

Once a list of potential pairs has been reduced by the above screening techniques, each pair must be individually tested for true compatibility.

TALON generates a miter circuit and then calls an ATPG tool, ATALANTA [7], for transformation verification.

Fig. 6 shows a generic miter circuit. The original and transformed circuits are treated as "black boxes." A network of XOR gates is used to compare each output. Any discrepancy between the two circuits will show up as a logic one at the output of one or more of the XOR gates, which will be passed on to the final output.

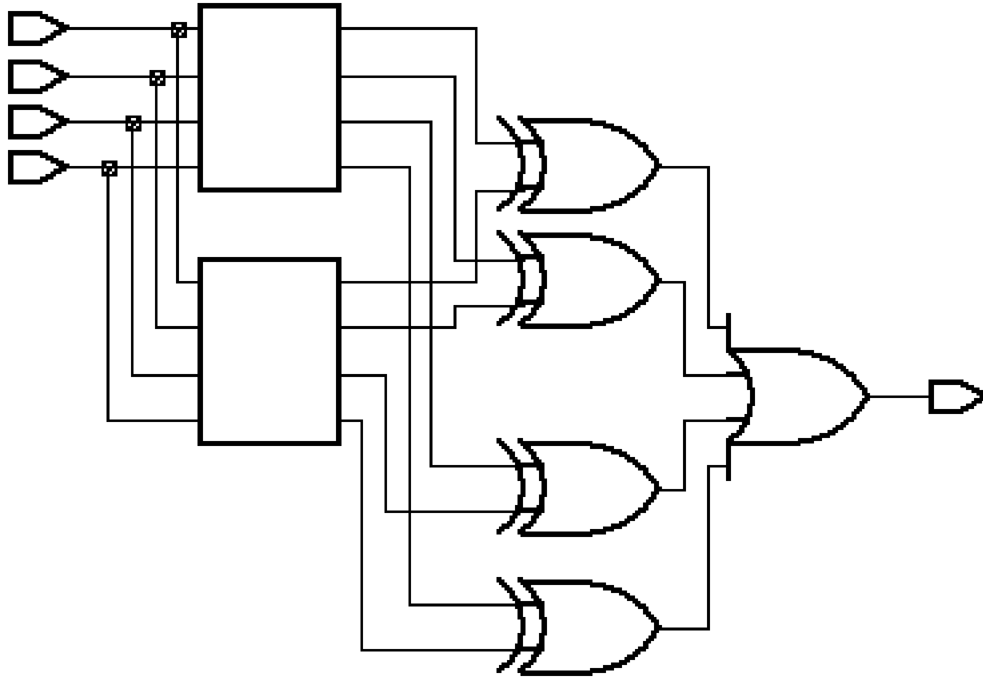

Fig. 6. Miter circuit for comparing two netlists

Testing the final output for stuck-at zero reveals if the modified circuit is functionally equivalent to the original. If a stuck-at zero fault could be detected at the output, then there must be some vector that would cause a fault-free circuit to output a logic one. If a fault-free circuit can output a logic one, then one or more of the output pairs can be set to conflicting values, and the two circuits are not functionally equivalent.

The first version of TALON would generate miter circuits of the type shown in Fig. 6. A drawback to this approach is that the topology that must be covered to find an output fault may be very large: it is, in fact, the entire device under test. This led to many good transformations being discarded when ATALANTA exceeded the set backtracking limit before determining if the output fault could be detected. Increasing the backtracking limit proved impractical, as even limits of several thousand backtracks failed to produce results, while using vast amounts of computer time.

Modifying the miter circuit to share all gates not in the fanout cone of the target gate provided only marginal improvement. The miter was then changed to XOR together the gate pair rather than the primary outputs of the original and modified circuits. Only the output of this XOR gate needs to be checked for stuck at zero detectability to determine if merging the gates would produce a functionally-equivalent circuit.

The sequence for testing pairs is illustrated in Figs. 7 through 9. In Fig. 7, two gates have been identified as a potential pair. In Fig. 8, a miter circuit is built. This circuit is tested to determine if the output of the XOR gate can be set to logic one. If the search space is exhausted without finding any input vector that can set this node high, the transformation is valid and the circuit of Fig. 9 is built.

If there is a vector that can set the XOR gate's output to logic one, then the pair is not a true pair. It is discarded and the next pair is tested.

C. Implicant Search

Once the naturally-occuring pairs in a netlist have been processed and the circuit reduced as far as the previously-described techniques can take it, TALON begins implication optimization.

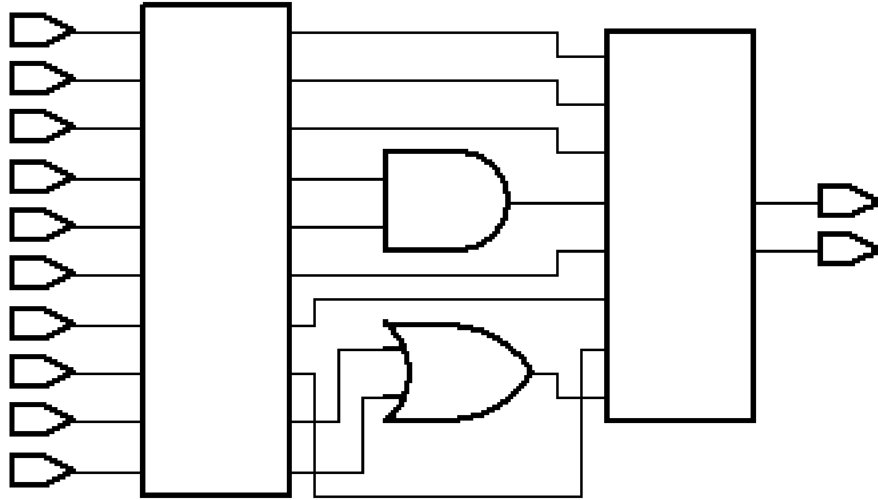

Fig. 7. Potentially-compatible gates in a larger circuit. The AND and OR gates have been identified as potentially-compatible.

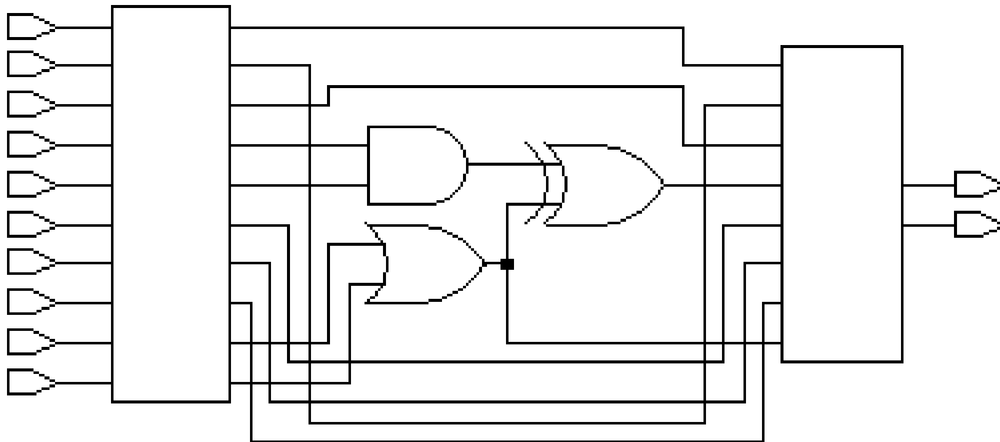

Fig. 8. Mitering a potential pair. An XOR gate is added to miter the potential pair.

The first step of implication optimization is to find pairs that form implicant functions. Though the procedure is similar to that of finding potentially-compatible pairs, the details of implementation are different.

Using one line (a line corresponds to a circuit node) from the fault detection matrix and another from the binary value matrix, the netlist is searched for pairs that would allow selected Boolean functions to be inserted without changing the circuit's functionality. The current TALON implimentation uses only AND and OR implication though, as noted in Chapter Five, other Boolean functions could be used.

A potential implicant pair is found when, for every vector that leads to a fault being detected at the target node, the binary values of some other node allow the target node to be replaced by an AND or OR function of the target node and some other node.

Table IV illustrates the process of finding implicant functions. The fault dictionary entries for one gate are compared to the binary values of another.

Gates A and B form an AND implicant function. Detecting a stuck-at zero fault implies that the binary value of the target node is logic one. Thus an implicant AND function can be formed from with a node that is also set to logic one whenever stuck-at zero is detected. This node can take on any value when no fault is detected or stuck-at one would be detected.

Gates C and D form an OR implicant function. In this case, the implicant function requires that the second node not conflict with the target node when stuck-at one faults are detected.

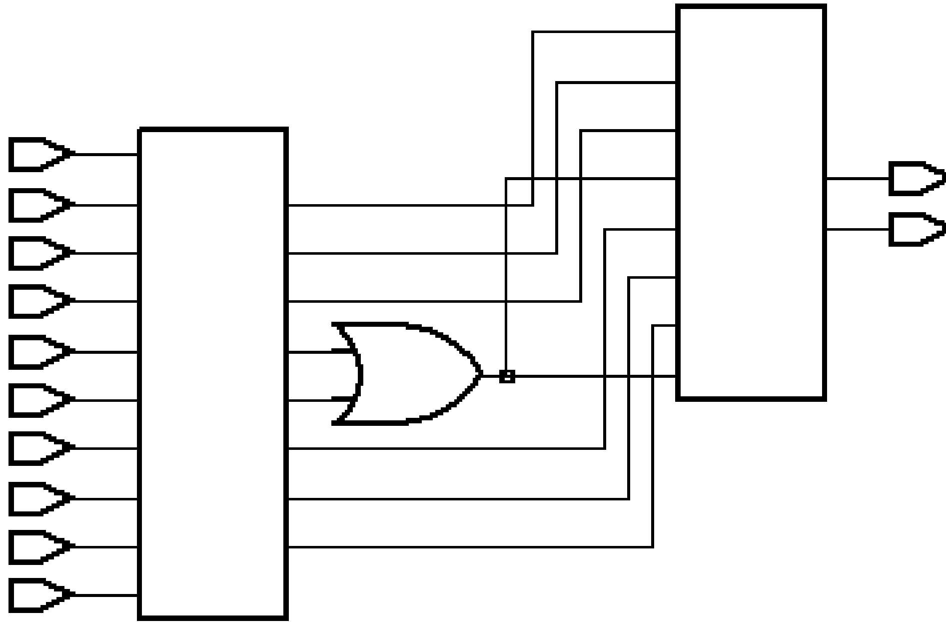

Fig. 9. Reduced circuit after mitering. If the miter reveals that the two are compatible, the target gate may be eliminated, along with any gates that only drive the target gate.

TABLE IV
IMPLICANT TABLE EXAMPLE. SAMPLE LINES FROM A FAULT DICTIONARY AND A BINARY VALUE MATRIX USED TO FIND IMPLICANT FUNCTIONS.

Gate A Fault Detection	1	N	N	0	N	N	1	0
Gate B Binary Values	1	0	0	1	1	1	0	1
Gate C Fault Detection	1	N	N	0	N	N	1	0
Gate D Binary Values	0	1	0	0	1	0	0	1

After a list of potential implicant pairs is formed, it is screened with the same tests used for potentially-compatible pairs.

D. Implicant Testing

Once a list of potential implicant pairs has been generated, they are tested with the same miter procedure as is used to test potentially-compatible pairs. If the XOR of the implicant gate and the target node can be set to logic one, the pair is immediately discarded. If no input vector exists that can cause that to happen, a netlist is formed with the implicant function and logic optimization is attempted on this resultant netlist.

Again using Fig. 7 as a starting point, the procedure for forming implicant functions is shown if Figures 10 and 11.

In Fig. 7, the AND and OR gates have been identified as a potential AND implicant pair. To test the viability of the implicant function, first the implicant gate is inserted into the circuit, as shown in Fig. 10.

An XOR miter gate is then added as shown in Fig. 11. As with circuit reduction mitering, if no vector exists that can cause a stuck-at zero fault to be detected at the output of the XOR gate, then the implicant function can be inserted without changing the circuit's functionality. Multi-level logic minimization is then performed on the circuit modified to include the implicant function.

E. Redundancy and Compatibility Optimization on Modified Netlist

Upon finding an implicant function that may be inserted into the design without changing the functionality, compatibility pairs are generated and tested as before. Transformations that lead to an overall reduction in the number of circuit connections are kept. Those that leave the netlist with the same or a larger number of connections are discarded. (A *connection* is defined as an input to a multi-input gate.)

Each time an area reduction is made, the implicant list is regenerated. Implicant pairs that have already been tried

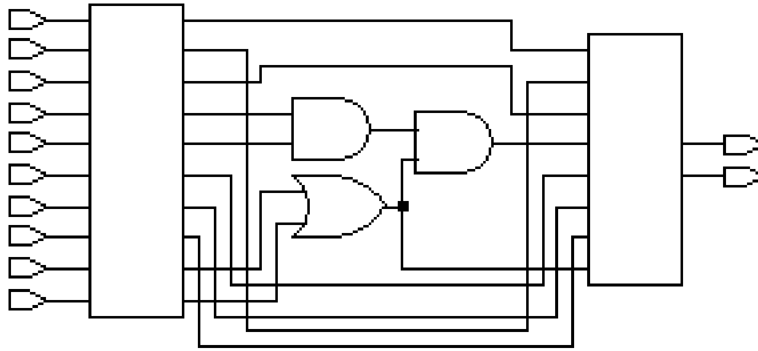

Fig. 10. AND implicant function.

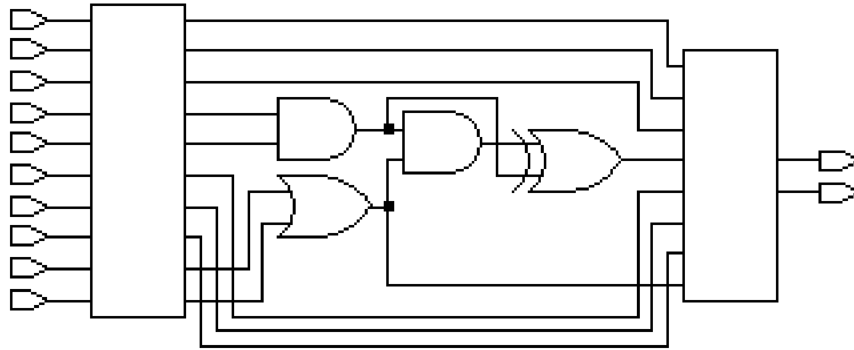

Fig. 11. AND implicant miter circuit.

are filtered out, along with all the other filtering described above. This may lead to some possible reductions being missed and makes optimization results more dependent on the order in which pairs are listed.

IV. Experimental Results

TALON was used to reduce selected ISCAS '85 combinational benchmark circuits. Only the smallest of these circuits were used (c432, c499 and c880), since TALON, as implemented today, runs slowly. Ease of programming, rather than execution speed, was the driving factor behind the current implementation of the TALON algorithm. Possible performance upgrades are suggested in the following chapter.

Nevertheless, TALON does reduce these circuits more than any other published process. When working in conjunction with proprietary commercial logic minimization tools, TALON produces still better results, suggestion a symbiotic relationship between the TALON algorithm and the Boolean arithmetic approach used by others.

A. Data Interpretation

TALON is purely a multi-level combinational logic minimization process. It does no technology mapping. Results are presented in terms of two-input gates. Both the number of connections in the resultant reduced netlists and the number of connections eliminated are provided here.

Comparisons are made to the number of connections eliminated by another academic tool, HANNIBAL [4]. This gives a significant advantage to HANNIBAL, since gate consolidation reduces the number of connections without performing any logic minimization. Just how much reduction is gained from technology mapping is a function of the library used. Data presented here also show the results of simply consolidating these circuits into a library containing the basic Boolean gates without restrictions on the number of inputs a gate can have. This library contains no complex (AND-

OR- INVERT, for example) cells. The library used for HANNIBAL does contain some complex cells, but limits gate widths to four inputs.

Despite this unlevel playing field, TALON still produces better optimization results in most cases.

Since both true and inverted phases of logic signals are available in some programmable logic arrays, but other technologies require additional resources to provide inverted signals, both connection data and the total number of inputs to all gates, including single-input gates, are presented in the following tables. The number of literals is not provided, as a search of technical literature revealed no clear and consistent definition for that metric.

B. Minimization Data

Tables in this subsection show area reduction data obtained with TALON and with other systems. Most experimentation was done with c432, though c499 and c880 were also optimized using TALON.

Before optimization was started, the ISCAS '85 netlists were decomposed into two-input gates. This accounts for the differences between the number of connections in the original netlist presented here and in [4]. Connection data for the original ISCAS circuits using multi-input gates are presented along with data for HANNIBAL connection reductions.

B.1 C432

Minimization data for c432 are presented in Table V.

Circuit types containing a "$\rightarrow$" symbol indicate that the first optimization procedure was used as a preprocessor for the second.

Table entries marked with an asterisk (*) were taken from [4] without verification by the authors. Complex, multi-input gates were used in these netlists. Also the authors can not establish with certainty that the connection counting methodology used was identical to that used for other tabular entries presented here. Gate input count data for these netlists are not available. The number of connections eliminated for these entries is in reference to ISCAS benchmark circuits before reduction to all two-input gates. It is not possible from published data to determine how many connections were eliminated due to technology mapping and how many due to logic minimization.

All rows below the double line in each of the following tables include technology mapping to libraries containing multi-input complex gates. The libraries used by HANNIBAL and Brand S are not necessarily identical.

Columns beginning with "Δ" indicate the reduction obtained through use of the associated methodology.

"Brand S" refers to a commercial logic synthesis tool. Although this tool is justifiably famous for its technology mapping capability, it was restricted to a library of only the basic Boolean functions (AND, OR, NAND, NOR, XOR, NOT and BUF). Results are presented with these gates decomposed to functions of two inputs.

TABLE V

C432 OPTIMIZATION DATA.

Circuit	Connections	Δ Connections	Gate Inputs	Δ Inputs
Original (2-I)	352	-	392	-
TALON	306	46	340	52
Brand S	298	54	323	69
Brand S $\rightarrow$ TALON	288	64	311	81
TALON $\rightarrow$ Brand S	274	78	295	97
Original (Multi-I)	296	-	336	-
HANNIBAL	207*	89	N/A	N/A
Brand S	212	84	244	92
TALON $\rightarrow$ Brand S	199	97	227	109

Although TALON proved capable of providing competitive minimization results by itself, the best results were obtained by using TALON as a preprocessor for a tool that reduces circuit size through algebraic manipulation. Results presented here show the complimentary nature of these algorithms.

B.2 C499

Minimization data for c432 are presented in Table VI. For this benchmark circuit, TALON was used as a preprocessor for Brand S, but not the other way around. Although further experimentation would be necessary before drawing any categorical conclusions, data gathered for c432 indicate that this is the method that gives the best results. If data gathered for c432 indicates a general trend, optimizing first with Brand S does not lead to as many reductions as does using that tool last.

TABLE VI
C499 OPTIMIZATION DATA.

Circuit	Connections	Δ Connections	Gate Inputs	Δ Inputs
Original (2-I)	412	-	452	-
TALON	366	46	390	62
Brand S	356	56	364	88
TALON → Brand S	330	82	342	110
Original (Multi-I)	368	-	408	-
HANNIBAL	348*	20	N/A	N/A
Brand S	326	42	336	72
TALON → Brand S	324	44	333	75

TABLE VII
C880 OPTIMIZATION DATA.

Circuit	Connections	Δ Connections	Gate Inputs	Δ Inputs
Original (2-I)	692	-	781	-
TALON	618	74	650	131
Brand S	578	114	598	183
TALON → Brand S	538	154	560	221
Original (Multi-I)	640	-	729	-
HANNIBAL	578*	62	N/A	N/A
Brand S	409	231	433	291
TALON → Brand S	402	238	429	295

B.3 C880

Minimization data for c432 are presented in Table VII. As with c499, time contraints prevented using Brand S as a preprocessor for TALON, though TALON was used as a preprocessor for Brand S and as a stand-alone minimization procedure.

C. Technology Mapping

To give an indication of the effects of technology mapping on gate input and connection statistics, data are presented in Table VIII showing

1: the original ISCAS '85 circuits
2: ISCAS '85 circuits reduced to two-input gates
3: These same circuits consolidated into a library permitting arbitrarily wide gates, but no complex gates.

TABLE VIII
EFFECTS OF GATE CONSOLIDATION ON CONNECTION AND GATE COUNTS.

Circuit	Connections	Gate Inputs
c432 Original	296	336
c432 (2-I)	352	392
c432 Consolidated	296	336
c499 Original	368	408
c499 (2-I)	412	452
c499 Consolidated	312	352
c880 Original	640	729
c880 (2-I)	692	781
c880 Consolidated	640	729

The reductions shown here, as much as 22%, do not represent the elimination of a single connection through logic minimization. A richer library, one including complex gates, multiplexors, and decoders, could be expected to provide still better results.

Consolidating gates in c499 leads to smaller counts than the original due to the employment of wide XOR gates in the

consolidated netlist. This single library addition leads to a 15% reduction in the number of connections in that netlist.

V. Conclusions and Future Work

TALON has proven itself to be a powerful system for multi-level logic optimization. Nevertheless, it may still be leaving some opportunities unexplored, it terms of possible area reductions.

The current implementation certainly leaves a lot to be desired in terms of execution speed. Some ideas on improving performance are outlined below.

A. Profiling

TALON is slow. Running a profiler to determine where the most time-consuming operations are could be a profitable step. Several of the following subsections deal with execution speed, but, without knowing exactly what is consuming all the time, they are speculative.

B. Recoding

Execution speed could certainly be enhanced by recoding TALON into a single compiled executable, rather than dozens of interpreted modules, each of which needs to be fetched on each loop.

C. Outer Loop Limits

Some additional methods of prescreening implicant pairs to cut down on the number of outer loops may exist. Finding more predictors of success or failure would cut down on the number of loops that are run that lead to no reduction in circuit size.

A small speed improvement might be possible by moving the distance screening to the initial pairup, rather than as a post-processing routine after doing an N^2 pair matching operation.

D. Implicant Functions

Data gathered here used AND and OR implicant functions. That still leaves a lot of possible functions unexplored.

For two nodes, A, and B, besides the degenerate cases of logic zero, logic one, A and B, other potential implicant functions include:

A NAND B
A NOR B
A XOR B
A XNOR B
A' AND B
A AND B'
A' OR B
A OR B'
etc.

XOR implications for logic optimization were used in [8], indicating that this is an area that bears more scrutiny.

E. Logical Proximity

Screening is done based on topological proximity. It may be possible to screen out pairs based on logical proximity, on how close two potential pairs match. No differentiation is made between potential pairs that have only a single observability mismatch and ones that have dozens. Grading the potential pairs on how close they match, and perhaps discarding pairs that exceed some threshold of mismatches, could be worth exploring.

F. Vector Generation

Potential pairs are found through fault simulation, which uses some previously-generated vector set.

An ideal vector set would screen out all pairs that are not truly compatible or are not true implicant pairs. Much work could be done on generating vector sets that approach this ideal, yet remain compact enough to make fault dictionary analysis viable for circuit minimization.

References

[1] S. Chang, M. Marek-Sadowska and K. Cheng, "Perturb and Simplify: Multilevel Boolean Network Optimizer," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* , vol. 76, pp. 1494–1504, December 1996.
[2] M. Grimaila, "TAMUFS272," private communication, Department of Electrical Engineering, Texas A&M University, July 1998
[3] M. Abramovici, M. Breuer and A. Friedman, *Digital Systems Testing and Testable Designs, Revised Edition*, New York, New York: IEEE Press, 1990.

[4] W. Kunz D. Stoffel and P. R. Menon, "Logic Optimization and Equivalence Checking by Implication Analysis," *IEEE Transactions on Computer Aided Design of Integrated Circuits and Systems*, vol. 16, pp. 266–281, March 1997.

[5] W. Kunz and D. K. Pradhan, "Recursive Learning: A New Implication Technique for Efficient Solutions to CAD Problems–Test, Verification and Optimization," *IEEE Transactions on Computer Aided Design of Integrated Circuits and Systems*, vol. 13, pp. 1143–1157, September 1994.

[6] F. Brglez and H. Fujiwara, "A Neutral Netlist of 10 Combinational Benchmark Circuits and a Target Translator in Fortran," in *IEEE Int. Symp. Circuits Syst*, Kyoto, Japan: IEEE Press, June 1985.

[7] H. K. Lee and D. S. Ha, "On the Generation of Test Patterns for Combinational Circuits," Technical Report No. 12_93, Dep't of Electrical Eng., Virginia Polytechnic Institute and State University, Blacksburg, Virginia.

[8] M. Chatterjee, D. Pradhan and W. Kunz, "LOT: Optimization with Testability– New Transformations Using Recursive Learning," *Proc. Int. Conf. Computer-Aided Design*, San Jose, California, Nov. 1995 pp. 318–325.