

REFERENCES

- [1] A. Baratz, "Algorithms for integrated circuit signal routing," Ph.D. dissertation, Dep. of Elec. Eng. and Comp. Sci., MIT, Cambridge, MA, 1981.
- [2] D. G. Boyer, "Symbolic layout compaction review," in *Proc. 25th Design Automation Conf.*, June 1988, pp. 383-389.
- [3] G. Cheng and A. Despain, "Fang: A joiner for compacted cells," in *VLSI 89*, 1989, pp. 455-463.
- [4] E. Horowitz and S. Sahni, *Fundamentals of Data Structures in Pascal*, 4th ed. New York: Freeman, 1994.
- [5] C. E. Leiserson and R. Y. Pinter, "Optimal placement for river routing," *SIAM J. Comput.*, vol. 3, no. 3, pp. 447-462, Aug. 1983.
- [6] A. Lim, "Efficient algorithms for CAD in VLSI," Ph.D. dissertation, Dep. of Comp. Sci., Univ. of Minnesota, Minneapolis, 1992.
- [7] A. Lim, S. Cheng, and S. Sahni, "Optimal joining of compacted cells," *IEEE Trans. Comput.*, vol. 42, no. 5, pp. 597-607, May 1993.
- [8] E. M. McCreight, "Priority search trees," *SIAM J. Comput.*, vol. 14, no. 2, pp. 257-276, May 1985.
- [9] A. Mirzaian, "River routing in VLSI," *J. Comput. and Syst. Sci.*, vol. 34, no. 1, pp. 43-54, 1987.
- [10] R. Y. Pinter, "On routing two-point nets across a channel," in *Proc. 19th Design Automation Conf.*, June 1982, pp. 894-902.
- [11] —, "River routing: Methodology and analysis," *Third Caltech Conference on VLSI*, R. Bryant, Ed. Rockville, MD: Computer Science, Mar. 1983, pp. 141-163.
- [12] F. P. Preparata and W. Lipski, Jr., "Optimal three-layer channel routing," *IEEE Trans. Comput.*, vol. C-33, no. 5, pp. 427-437, May 1984.
- [13] M. Tompa, "An optimal solution to a wire-routing problem," *J. Comput. and Syst. Sci.*, vol. 23, no. 2, pp. 127-150, May 1981.
- [14] N. Weste, "Virtual grid symbolic layout," in *Proc. 18th Design Automation Conf.*, June 1981, pp. 225-233.

Combinational and Sequential Logic Optimization by Redundancy Addition and Removal

Luis A. Entrena and Kwang-Ting Cheng

Abstract—This paper presents a method for multilevel logic optimization for combinational and synchronous sequential circuits. The circuits are optimized through iterative addition and removal of redundancies. Adding redundant wires to a circuit may cause one or many existing irredundant wires and/or gates to become redundant. If the amount of added redundancies is less than the amount of created redundancies, the transformation of adding followed by removing redundancies will result in a smaller circuit. Based upon the Automatic Test Pattern Generation (ATPG) techniques, the proposed method can efficiently identify those wires for addition that would create more redundancies elsewhere in the network. Experiments on ISCAS-85 combinational benchmark circuits show that best results are obtained for most of them. For sequential circuits, experimental results on MCNC FSM benchmarks and ISCAS-89 sequential benchmark circuits show that a significant amount of area reduction can be achieved beyond combinational optimization and sequential redundancy removal.

I. INTRODUCTION

Many multilevel combinational optimization techniques that improve the results of simple algebraic methods [1] have been proposed

Manuscript received July 7, 1993; revised April 18, 1995. This paper was recommended by Associate Editor F. Somenzi.

L. A. Entrena is with TGI, Plaza Marques de Salamanca 3, 28006 Madrid, Spain.

K.-T. Cheng is with the Department of Electrical and Computer Engineering, University of California, Santa Barbara, CA 93106 USA.

IEEE Log Number 9412527.

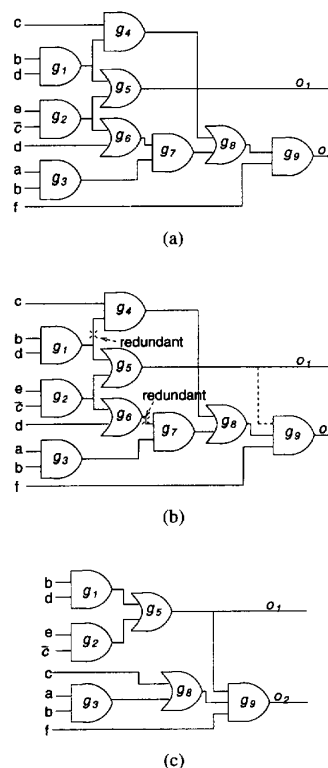


Fig. 1. A combinational example. (a) An irredundant circuit. (b) Adding a redundant connection. (c) The final irredundant circuit.

in the past few years [2]–[6]. The key to these optimization techniques is the use of don't cares. These techniques have also been extended for sequential logic optimization [7], [8]. In [8], a don't-care based minimization technique for sequential circuits is proposed that extends the concept of permissible functions for flip-flops. These methods, however, cannot handle circuits with feedback loops. They cut those feedback loops and treat the corresponding signals as primary inputs and primary outputs.

Redundancy removal has achieved certain success for improving testability and optimizing logic for large multilevel combinational and sequential networks [9], [10]. Efficient test generation technique like SOCRATES [11] used for redundancy identification results in fast execution time for large networks.

In this paper, we present a method for optimizing combinational and synchronous sequential logic networks. The new method is based on Automatic Test Pattern Generation (ATPG) techniques. In this new method, named *Redundancy Addition and Removal*, the network is optimized through iterative addition and removal of redundant connections. Adding redundant wires to a network may cause one or many existing irredundant wires and/or gates to become redundant. If the amount of added redundancies is less than the amount of created redundancies, the transformation of adding followed by removing redundancies will result in a smaller network.

Example 1: Consider the circuit given in Fig. 1(a). This circuit is irredundant. If we add a connection from the output of gate g_5 to the input of gate g_9 (shown as a dashed line in Fig. 1(b)), the functionality

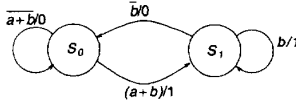


Fig. 2. A two-state machine.

of the circuit does not change. In other words, the added connection is redundant. However, the addition of the connection causes two originally irredundant wires becoming redundant as shown in Fig. 1(b). After removing these two wires and associated gates that either become floating (g_6) or have a single fanin (g_4 and g_7), the circuit can be greatly optimized as shown in Fig. 1(c).

From one viewpoint, the suggested technique is a generalization of Boolean Resubstitution [3]. Boolean Resubstitution in the BOLD system [3] and its improvements using permissible functions and Ordered Binary Decision Diagrams (OBDD) [12], [13] relies on: 1) an adaptation of the ESPRESSO algorithm suite to the context of a multilevel Boolean network; and 2) a method for multilevel tautology checking to verify the correctness of the transformations. These two parts work independently of each other. In our approach, instead, data is collected during the verification phase in the form of mandatory assignments that is used to guide the search for efficient candidates for redundancy addition. This gives our new approach a significant advantage over previous work based on the Boolean Resubstitution paradigm and allows the efficient optimization of much larger networks.

The proposed technique can also be viewed as a generalization of redundancy removal. Contrast to existing sequential optimization approaches, the new method allows Boolean minimization for multilevel synchronous sequential circuits with feedbacks. Also, by adding and removing redundancy, it is possible to migrate from one state assignment to another that can be implemented with less wires/gates and/or flip-flops.

Example 2: Fig. 2 shows the state graph of a two-state machine. The machine has two inputs, a and b , and one output. An one-hot coded, combinational optimized implementation is given in Fig. 3(a). The output is simply the next state line y_1 . If we add $\overline{a}$ to the input of gate g_1 , the sequential behavior of the circuit will not change as shown in Fig. 3(b). However, signal x_2 becomes sequentially redundant after this addition as shown in Fig. 3(c). After removing x_2 and its fanins which become floating, the circuit has only one flip-flop, as shown in Fig. 3(d), and is a minimal-bit encoded machine.

The problem of identifying connections that can be added or removed without affecting the I/O behavior of the network can be converted into a test generation problem for a stuck-at fault [14]. In [14], a fault model, called the connection fault model, is introduced. A connection fault, characterized by a triple (source gate, destination gate and polarity), causes an extra connection (inverted or noninverted) from the output of the source gate to the input of the destination gate. Determining whether a connection can be added without changing the functionality of the network is equivalent to determining whether a connection fault is redundant. By proper modeling, redundancy identification for connection faults can be converted into the problem of redundancy identification for stuck-at faults. Two issues need to be resolved in this approach. 1) For a network of N nodes, there are $O(N^2)$ connections to be examined. Usually, only a very small percentage of them is redundant and even a smaller percentage of them would create other redundancies. It would be too complex to look at all of these $O(N^2)$ connections explicitly for determining whether they should be added or not. 2) For certain stuck-at and connection faults, it is very time consuming to determine whether the fault is redundant. In this paper, we propose techniques that overcome

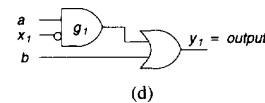
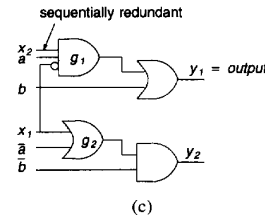
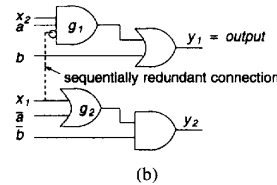
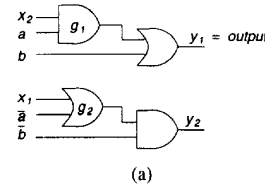


Fig. 3. (a) An one-hot encoded implementation ($S_0 = 01$, $S_1 = 10$). (b) Adding a sequentially redundant connection. (c) Flip-flop 2 becomes sequentially redundant. (d) Final minimized network.

these two problems. Based on the concept of *mandatory assignments*, a fast redundancy identification procedure (for both stuck-at and connection faults) that does not require backtracking is proposed. This procedure is extended using the concept of *Recursive Learning* [15] for computing all mandatory assignments. For a given fault, the mandatory assignments are those assignments that are required for a test to exist. Implication of the mandatory assignments throughout the network is performed to determine whether they can be consistently applied. If any inconsistency is detected, no test is possible and the fault is redundant. If all mandatory assignments are consistent and the fault is not redundant, our procedure will identify a set of connections that can be added to force the set of mandatory assignments to become inconsistent and thus force the fault to become redundant. These connections are then sorted and processed according to the amount of redundant faults that would be created by its addition. Using this idea, the number of connections that are explicitly considered is very small and the ones that create more redundancies can be identified in a very efficient manner.

Section II gives the background definitions for terms used in the later sections. Section III describes a redundancy identification procedure based on the concept of mandatory assignments. Section IV shows how to identify candidate connections to add that will create redundant faults. Section V shows the overall algorithm for logic optimization. Section VI gives the experimental results.

II. BACKGROUND DEFINITIONS

A *combinational circuit* is a *direct acyclic graph (DAG)*. The vertices of the DAG are the primary inputs, the primary outputs and

the gates of the circuit. We use the term *node* to refer to either a primary input or a primary output or a functional gate with more than one input, i.e., inverters are not considered nodes.

A *connection* is the edge connecting two nodes and it is characterized by a triple (S, D, P) , where S is the source node, D is the destination node, and P is the polarity (1 for inverted and 0 for noninverted). A connection is combinational (sequentially) redundant if it can be removed without affecting the I/O behavior of the combinational (sequential) network.

A vertex A in a graph G is a *dominator* of another vertex V in G with respect to an output O if and only if every path from the dominated vertex V to the output O passes through the dominator vertex A . A dominator of a vertex V is an *absolute dominator* of V if and only if it is a dominator with respect to all reachable outputs. A vertex A is an absolute dominator of a connection (S, D, P) if it is an absolute dominator of the destination gate D .

A functional gate N in the circuit (it could be an AND, OR, NAND or NOR gate) is characterized by two values, the *controlling value* $Cont(N)$ and the *inversion value* $Inv(N)$. The value of an input is said to be controlling if it determines the value of the gate output regardless of the values of the other inputs; then the output value is $Cont(N) \oplus Inv(N)$. The controlling value is 1 for an OR or a NOR gate, and 0 for an AND or a NAND gate. The inversion value is 1 for a NAND or a NOR gate, and 0 for an AND or an OR gate. The inverse of the controlling value is often called the *sensitizing value* $Sens(N)$.

In the context of test generation for stuck-at faults, a *mandatory observation node* is a node that must be visited when propagating the fault effect to at least a primary output. For a gate input fault to node N , the absolute dominators of N are mandatory observation nodes. For a gate output fault at node N , the absolute dominators of N except N itself are mandatory observation nodes. The observability of the fault effect requires that all side inputs to the mandatory observation nodes have a sensitizing value (*unique sensitization*).

III. REDUNDANCY IDENTIFICATION BASED ON MANDATORY ASSIGNMENTS

Test generators for stuck-at faults are often used for redundancy identification. If no test exists for a given fault, then the fault is redundant and the circuit can be simplified by removing at least a gate or a gate input. However, most ATPG tools are focused on finding test vectors for the maximum number of faults and maximizing the fault coverage, rather than on identifying redundancy. This can introduce some inefficiency if the goal is identifying redundancy.

Our redundancy identification procedure is based on test generation techniques. It is focused on: 1) identification of redundancy—rather than test pattern generation; and 2) fast execution. The first goal is achieved by searching for inconsistency rather than for test vectors. The second one is achieved by completely avoiding backtracking at all.

The assignments made during a search for a test can be divided into two categories: *mandatory assignments* and *optional assignments*. Mandatory assignments are those which are required for a test to exist and must be satisfied by any test vector. The mandatory assignments can be further classified into mandatory control assignments and mandatory observation assignments. The mandatory control assignments are those that activate the fault. The mandatory observation assignments are those that sensitize the fault to any output. Note that the mandatory control assignments for a connection fault $(S, D, 0)$, are those that will set S to the noncontrolling value of gate D . For example, if D is an AND gate, the mandatory control assignments are those to set S to 1. Similarly, the mandatory control assignments for

a connection fault $(S, D, 1)$, are those that set S to the controlling value of gate D .

In order to identify redundancy, only the mandatory assignments need to be considered. For a given fault f , we compute the *set of mandatory assignments*, $SMA(f)$. The justification of the mandatory assignments may require some nodes to take some mandatory values. These mandatory values are computed via implication of the mandatory assignments in $SMA(f)$, until no further implication is possible or any inconsistency is detected. If the mandatory assignments cannot be consistently justified, then the fault is redundant. In that case we say that the set of mandatory assignments for the fault f is *incompatible*. The process of proving the incompatibility of the set of mandatory assignments for the fault f is called *redundancy testing* for the fault f .

The identification of the mandatory assignments is crucial for our technique. Techniques that allow the identification of more mandatory assignments and improve the implication process are desirable because of two reasons: 1) more redundancy can be identified; and (2) more candidate connections to be added can be generated. However, the problem of finding all the mandatory assignments is NP-complete and cannot be resolved within a reasonable time for some hard faults. An important number of mandatory assignments are found in a very efficient manner with the use of the concepts of static [16] and dynamic absolute dominators [11] and the nine-valued model for test generation [17]. Using these concepts we have built a very fast redundancy identification procedure that is able to identify a great deal of redundancy. Recently, Kunz and Pradhan [15] proposed a recursive method for finding all the mandatory assignments. Their experiments showed that the maximum recursion depth required is low for practical cases. However, the time complexity of this method is exponential in the maximum depth of recursion and cannot be used for large networks. We have implemented the recursive learning method in our prototype system and use it as an option for computing the mandatory assignments.

A. Redundancy Identification for Sequential Circuits

This technique is extended for synchronous sequential circuits. First, the mandatory assignments for exciting the fault site from the primary inputs and present state lines and for propagating the fault effect to any primary output or next state line are computed. Implication of these mandatory assignments is performed within one time frame, called *time frame 0*. This is equivalent to applying the above procedure to the combinational portion of the sequential circuit. If the set of mandatory assignments is found inconsistent at this point, the fault is combinational redundant. Otherwise we continue to justify the mandatory assignments at the present state lines in the previous time frames. Also, if no primary output is reachable from the fault site at time frame 0, the fault must be propagated into the next time frames until some primary output is reachable. New mandatory assignments may appear at the next time frames by applying the unique sensitization condition, which requires all side inputs to the mandatory observation nodes to be sensitizing.

Implication of the mandatory assignments may occur across time frames. During justification at a given time frame new mandatory assignments may be implied at the next state lines that can be transferred to the present state lines of the next time frame. During propagation, new mandatory assignments may appear at the present state lines that must be justified in the previous time frame.

If any inconsistency is found during implication of the mandatory assignments then the fault is untestable. However, redundancy and undetectability are not equivalent for sequential circuits. For circuits that have a global reset state for both fault-free and faulty circuits, the absence of a test is equivalent to the existence of redundancy. For circuits without a global reset state the absence of a test still implies

redundancy if the fault cannot be propagated to a primary output independently of the initial state or if the required initial state cannot be activated in the fault-free circuit [9]. Two types of inconsistency are possible for sequential circuits: 1) local inconsistency: two different mandatory assignments made on the same node; and 2) cyclic inconsistency: a repeated pattern of mandatory assignments to be justified or propagated from time frame to time frame. The following theorems allow the detection of cyclic inconsistency.

Theorem 1 (Activation): Let NSS be the set of next state lines and PSS be the set of present state lines. Assume the target fault f is excited in time frame t_0 . In order to excite f in t_0 , set $A(f, t_0)$ of mandatory assignments for a subset of PSS for t_0 are obtained (is equivalent to say, that machine has to be in a subset S_0 of states to excite f). These mandatory assignments for PSS in t_0 are also mandatory assignments for NSS in t_{-1} which itself may lead to mandatory assignments for PSS in t_{-1} and so on. Each set of mandatory assignments at the PSS of a time frame with index $i < 0$ corresponds to a subset of states S_i the machine has to be in at time frame i , in order to excite the fault f in t_0 . If there exists an i such that $S_i \subseteq S_0$, then the fault is untestable.

Proof: By definition, the states in S_0 are only reachable from the states in S_i , $i < 0$. If there exists an i such that $S_i \subseteq S_0$, the states in S_0 are only reachable from a subset of S_0 and cannot be reached from states not in S_0 . Therefore, the states in S_0 cannot be reached starting from an unknown state and, thus, fault f is untestable.

Theorem 2 (Propagation): In order to excite f and propagate the fault effect to NSS in t_0 , sets $A_g(f, t_0)$ and $A_f(f, t_0)$ of mandatory assignments for a subset of NSS for t_0 are obtained for the fault-free and faulty circuits respectively. These mandatory assignments for NSS in t_0 are also mandatory assignments for PSS in t_1 which itself may lead to mandatory assignments for NSS in t_1 and so on. Each set of mandatory assignments at the NSS of a time frame with index $i > 0$ corresponds to a subset of state-pairs S_{g_i}/S_{f_i} , the fault-free and faulty circuits have to reach at time frame i , in order to detect the fault f at some primary output. If there exists an i such that $S_{g_0} \subseteq S_{g_i}$ and $S_{f_0} \subseteq S_{f_i}$, then the fault is untestable.

Proof: By definition, the state-pairs in S_{g_i}/S_{f_i} , $i > 0$, are only reachable from the state-pairs in S_{g_0}/S_{f_0} . If there exists an i such that $S_{g_0} \subseteq S_{g_i}$ and $S_{f_0} \subseteq S_{f_i}$, the state-pairs in S_{g_i}/S_{f_i} are only reachable from a subset of S_{g_i}/S_{f_i} . Therefore, the state-pairs in S_{g_i}/S_{f_i} cannot be reached starting from an unknown state and, thus, fault f is untestable.

IV. IDENTIFICATION OF CANDIDATE CONNECTIONS FOR ADDITION

In this section we address the problem of selecting candidate connections for addition that lead to an efficient minimization of the network. The key idea of the proposed method is forcing the SMA to be incompatible. For an irredundant fault, the set of mandatory assignments is compatible. However, we can force this set not to be so by adding some connections or nodes to the network to create new mandatory assignments. In particular, this can be easily done by exploiting the unique sensitization condition.

Suppose that the mandatory value v was assigned to node N during redundancy testing for fault f . Also let A be a mandatory observation node for the fault f and let c be the controlling value of gate A . Then, by adding the connection $C(N, A, 0)$ if $v = c$ (or $C(N, A, 1)$ if $v \neq c$) the necessary assignment $N = \bar{v}$ is added to $SMA(f)$ by unique sensitization. This is inconsistent with the previous mandatory value of node N , therefore the fault f is made redundant by adding the connection C . If adding connection C does not change the network's functionality (that needs to be examined separately), then, by adding this connection, at least one connection somewhere else can be removed.

Example 3: Consider the circuit in Fig. 1(a) and the fault g_6 stuck-at-1. This fault is not redundant, therefore its SMA is compatible. The mandatory assignments are: 1) mandatory control assignment: $g_6 = 0$; 2) mandatory observation assignments: $g_3 = 1$, $g_4 = 0$, $f = 1$. From $g_6 = 0$ we get $d = 0$ and $g_2 = 0$. From $g_3 = 1$ we get $a = 1$ and $b = 1$. These further imply $g_1 = 0$ and $g_5 = 0$. Because g_9 is a mandatory observation node for the target fault (g_6 stuck-at-1) and g_5 has an implied value that is the controlling value for gate g_9 , it suggests the addition of connection $(g_5, g_9, 0)$. To verify that this connection can be added to the network without changing its I/O behavior, a new redundancy test is required. The mandatory assignments for this connection fault are: 1) mandatory control assignment: $g_5 = 0$; 2) mandatory observation assignments: $g_8 = 1$, $f = 1$. It can be easily shown that this set of mandatory assignments is incompatible. Therefore the connection $(g_5, g_9, 0)$ can be added and the connection $(g_6, g_7, 0)$ can be removed. Furthermore, the connection $(g_1, g_4, 0)$ becomes redundant in the presence of the added connection as well. The final circuit after adding and removing these redundancies is shown in Fig. 1(c).

For sequential circuits, if there are mandatory values implied in the previous time frame, it is possible to add connections across time frames by inserting flip-flops. For example, if node N in time frame $t - i$ has a mandatory value v for fault f , and A is a mandatory observation node for fault f in time frame t , it is possible to add connections from node N in time frame $t - i$ to gate A in time frame t by adding i extra flip-flops between them. Because the cost of an extra flip-flop is generally much higher than the cost of a simple gate, these connections should not be added unless an equivalent amount of flip-flops and/or gates/wires is removed elsewhere.

V. THE ALGORITHM

In this section we explain the way we put together the ideas described above for global logic optimization. Basically, the optimization of the network is achieved by minimizing the amount of logic that is required to implement the logic function at the output of each node. In this work, we consider the total connection count as an estimation of the area of the circuit. This measure can be viewed as equivalent for multilevel networks to the literal count, that is the usual figure of merit for two-level minimization algorithms. The *cost of a node* is defined as the number of connections in the fanout free region (FFR) of this node (or equivalently, the number of connections that are absolutely dominated by the node). The *cost of a fault* is defined as the number of connections that can be removed if the fault is redundant.

Fig. 4 shows the flow of the algorithm we used for logic optimization. The algorithm makes a loop through all the nodes in the network in a cost descending order. At each iteration the optimization of one node is addressed. A partial fault list is created for the connections in the FFR of this node (i.e., the connections in the input tree of the node not shared with other nodes) and redundancy testing is performed for every collapsed stuck-at fault in the fault list. If the SMA of any of these faults is incompatible, the fault is redundant and we proceed to remove it. Otherwise, the mandatory values at all nodes outside of the fanout cone of the target node that are not in its input tree are collected. Notice that the input tree of a node may extend across time frames for sequential circuits.

For every fault in the partial fault list, a table, called the *optimization table*, is created after the redundancy testing is completed. In this table, there is a row for each fault in the partial fault list for the target node, and a column for each candidate connection to be added to the target node, as suggested by the mandatory values collected previously during redundancy testing. If the fault corresponding to row r becomes redundant in the presence of the

```

For each node  $N$ 
{
  do {
    Generate a partial fault list  $L$  for node  $N$ 
    For each fault  $f$  in  $L$ 
      if  $f$  is redundant remove  $f$ ;

    Generate the optimization table  $OT$ ;

    If there is a column  $c$  in  $OT$  such that the corresponding connection  $C_c$  is redundant
    {
      Add  $C_c$ ;
      For each marked row  $r$  in column  $c$  of  $OT$ 
        if the fault  $f_r$  in row  $r$  is redundant
          remove  $f_r$ ;
    }
  } while the connection count for  $N$  is reduced;
}

```

Fig. 4. A one connection addition algorithm.

TABLE I
OPTIMIZATION TABLE FOR g_9

	$(g_5, g_9, 0)$	$(g_5, g_9, 1)$
g_8 stuck-at-1		
f stuck-at-1		
g_4 stuck-at-0		*
g_7 stuck-at-0		
c stuck-at-1		*
$g_1(g_4)$ stuck-at-1	*	
g_6 stuck-at-1	*	
g_3 stuck-at-1		
$g_2(g_6)$ stuck-at-0		*
$d(g_6)$ stuck-at-0		*
a stuck-at-1		
$b(g_3)$ stuck-at-1		

connection corresponding to column c , then the element in row r and column c in the optimization table is marked. At least one element for every column will be marked. More than one element may be marked in the same column, since adding one connection may make more than one fault redundant. Analogously, more than one element may be marked in the same row, as a given fault may become redundant by the addition of several candidate connections, one at a time. However, note that adding a connection does not mean that all the faults for the marked elements in its column can be removed, because after redundancy removal of one single gate or wire the SMA of the remaining faults has changed and the entire table must be recomputed.

Example: Consider the circuit in Fig. 1(a). The fault list for node g_9 is: g_8 stuck-at-1, f stuck-at-1, g_4 stuck-at-0, g_7 stuck-at-0, c stuck-at-1, $g_1(g_4)$ stuck-at-1, g_6 stuck-at-1, g_3 stuck-at-1, $g_2(g_6)$ stuck-at-0, $d(g_6)$ stuck-at-0, a stuck-at-1 and $b(g_3)$ stuck-at-1. The only node outside of the input and output cones of node g_9 is g_5 . Therefore, the optimization table has two columns, one for the connection $(g_5, g_9, 0)$ and another one for the connection $(g_5, g_9, 1)$. Table I shows the optimization table for the node g_9 .

Since the faults $g_1(g_4)$ stuck-at-1 and g_6 stuck-at-1 imply a mandatory value 0 at node g_5 , they can be made redundant by adding the connection $(g_5, g_9, 0)$. Similarly, the faults g_4 stuck-at-0, c stuck-at-1, $g_2(g_6)$ stuck-at-0 and $d(g_6)$ stuck-at-0 imply a mandatory value 1 at node g_5 and they can be made redundant by adding the connection $(g_5, g_9, 1)$. This is the meaning of the marked elements in the optimization table.

One candidate connection is then selected and redundancy testing is performed for this fault. If this connection fault is found irredundant, we select another one. Otherwise we add the redundant connection.

Then we can carry out redundancy removal for at least one of the faults marked in its column. For the rest of the faults marked, redundancy testing must be performed again. Several heuristics may be used in order to select a good candidate connection. We have tried the following:

Heuristic 1: The simplest heuristic consists in selecting the candidate connection that makes the fault of the highest cost become redundant. In case this fault is made redundant by several candidate connections, Heuristic 2 is applied.

Heuristic 2: For each column, sum the cost of all the faults in the marked rows and select the one with the highest total cost. In case the highest total cost is reached by several candidate connections, apply Heuristic 1.

Note that if after adding one candidate connection C only one single connection C' can be removed, no improvement is produced. In that case we will say that C is an *alternative connection* for C' . Selecting a particular connection out of all the possible alternative connections for a given connection C' has no influence in the optimization of the current target node, but may influence the optimization of other nodes. For example, by adding a connection from a single fanout node, we may prevent the removal of this node later on the optimization process. For this reason, we prefer connections from primary inputs or primary outputs to connections from functional nodes. Among the functional nodes, we select those with the highest fanout since they are less likely to be removed later. When no further optimization is possible for the target node, we select the best alternative for every connection.

A. Adding More Than One Connection

The algorithm discussed so far contemplates only the case of adding a single redundant connection and removing one or more connection/nodes. Allowing adding more than one redundant connections at a time and removing more connections/nodes than were added may result in a more powerful transformation. However, selecting an optimal set of connections/nodes to be added is a highly complex task, unless some restrictions are applied. In this section we describe a multiple connection addition approach. In this approach, all the connections added at a time share the same destination node.

This approach works as follows. Similarly to the one connection addition approach, a node is selected at every iteration and the optimization table is created. Then we do redundancy testing for each candidate connection to identify those that may be added to the network without changing its I/O behavior. However, no addition is performed when a connection is found redundant. Instead, the addition is deferred until all candidate connections have been tested. Note that after adding one redundant connection it is not necessary to do redundancy testing again for the candidate connections that were previously identified as redundant. This is due to the following. Let f be the fault associated to the addition of a given connection C to the target node and $SMA_0(f)$ its set of mandatory assignments. Let $SMA_1(f)$ be the set of mandatory assignments for f after one connection, different than C , is added to the target node. Clearly the addition of the latter connection introduces new mandatory assignments into $SMA_1(f)$. Because the only difference of the new circuit C_1 from the initial circuit C_0 is an extra input at the target node, we can claim $SMA_0(f) \subseteq SMA_1(f)$. Hence, if f was redundant in the initial circuit C_0 (i.e., $SMA_0(f)$ is not consistent), then $SMA_1(f)$ will not be consistent and, therefore, f is also redundant in circuit C_1 .

After all the connections are added, we enter the redundancy removal step. We create a partial fault list for all the connections that are dominated by the target node and apply redundancy removal.

TABLE II
RESULTS FOR COMBINATIONAL BENCHMARK CIRCUITS

Name	Initial			RAMBO_C				MIS-II #L	MIS-II+ RAMBO_C #L	Reduction (%)
	#G	#C	#L	#G	#C	#L	time			
C1355	378	859	562	374	837	546	67.1	550	550	0.0
C1908	442	1090	769	323	784	551	90	518	517	0.2
C2670	678	1827	1023	531	1520	816	150.5	741	733	1.1
C3540	878	2172	1658	727	1810	1331	1475.3	1253	1172	6.5
C432	125	380	270	78	271	207	6.2	196	176	10.2
C499	378	859	562	374	837	546	68.4	550	550	0.0
C5315	1751	3927	2425	1408	3201	1851	925.6	1740	1704	2.1
C6288	1888	3840	3313	1885	3834	3294	895	3313	3280	1.0
C7552	2177	4795	3087	1500	3385	2188	1744	2157	1914	11.3
C880	259	683	433	240	643	410	16.8	427	413	3.3
alu2	284	685	453	208	509	359	139	341	300	12.0
alu4	567	1320	855	415	1006	722	678.6	780	679	12.9
apex6	548	1447	835	498	1327	759	153.4	754	710	5.8
apex7	179	497	289	141	412	251	18.5	248	239	3.6
dalv	1847	4159	2610	808	2007	1344	5008.2	745	706	5.2
frg2	1111	3058	2005	560	1734	1157	1172.4	775	735	5.2
k2	278	3066	2928	262	2232	1996	1489.7	1054	1018	3.4
pair	1086	2866	1803	977	2594	1636	437.2	1606	1560	2.9
rot	448	1289	764	373	1093	662	71.3	664	630	5.1
term1	271	709	456	126	363	248	56.7	166	143	13.9
ttu2	180	503	324	108	300	191	13.8	206	175	15.0
vda	126	1482	1423	123	1222	1160	168.8	593	574	3.2
x1	222	694	445	154	503	333	36.1	298	290	2.7
x3	760	1951	1133	592	1547	985	336.2	773	739	5.0
x4	314	995	607	267	777	449	85.1	391	367	6.1
9symml	167	403	237	136	324	217	56.3	209	197	5.7

The partial fault list for redundancy removal is ordered according to:
1) original connections are placed before new connections; and 2) descending order of the fault cost. Redundancy testing is carried out for each fault in this fault list until no redundant faults left.

VI. APPLICATION TO TIMING OPTIMIZATION

With slight modification, the suggested technique can be used for timing optimization. The modifications affect the partial fault list generation, cost computation and scope of the main loop.

Suppose S is the source and D is the destination of the connection fault. D_{in_i} , $i = 1, 2, \dots, k$, are the existing fanins of gate D which has k fanins. The arrival time of a signal g is denoted as $Arr(g)$. In the step of partial fault list generation, we only consider those connection faults that have the following properties: 1) D is on critical paths; 2) $Arr(S)$ is not greater than the maximum of $Arr(D_{in_i})$, $i = 1, 2, \dots, k$. If such a fault is redundant and the corresponding connection is added, the timing of the network will not be degraded. If the addition of such redundant connection(s) cause any of the wires on critical paths to become redundant, the network's performance can thus be improved by removing the redundant critical net. In the step of determining the processing order of stuck-at faults in redundancy removal, the cost of a fault can be defined as the amount of speed improvement of the network if the signal is deleted. If there are many critical paths, the cost can be defined as the performance improvement of the critical paths that the stuck signal resides on. The cost of each fault in the stuck-at fault list should be incrementally recomputed and reordered each time a redundant stuck-at fault is identified and the network is modified. Finally, the main loop goes over all the nodes in the critical path, as we only consider connection addition to nodes in the critical path.

Since this timing optimization is performed at the technology independent level, in practical, it can be used before applying other technology dependent timing operations [18].

VII. EXPERIMENTAL RESULTS

In this section we present experimental results for both combinational and sequential benchmark circuits. Our prototype system is named RAMBO (Redundancy Addition and removal for Multilevel Boolean Optimization). There are two versions of the system: RAMBO_C (the combinational version) and RAMBO_S (the sequential version). We first compare the results of RAMBO_C for combinational benchmarks to those produced by MIS-II [1], [19]. All CPU times are in seconds on SUN SPARCStation 2. Also, we then show that sequential optimization with RAMBO_S produces a significant improvement in the flip-flop, gate and connection counts for many MCNC FSM benchmarks and ISCAS89 sequential benchmarks within an affordable extra runtime.

Table II shows the results for some of the MCNC combinational benchmark circuits. The column titled RAMBO_C gives the results after running RAMBO_C (without recursive learning) on the initial examples. This includes a preliminary redundancy removal step, a fast run with redundancy being added only from multiple fanout nodes and nonfunctional nodes, and a full minimization step. The gate count (#G), connection count (#C), literal count in factorized form (#L) are listed for the initial and final network, along with the amount of CPU seconds consumed on SUN SPARCStation 2. The column titled MIS-II shows the results obtained with MIS-II. For each example, we run *full_simplify* when possible followed by the simplification script *script2* described in [19]. (For some examples we could not run *full_simplify* because the BDD could not be built. For those examples, we ran *simplify -m nocomp* instead.) This script includes boolean operations and produces the best results for some circuits published up to date. In the column titled MIS-II+RAMBO_C we show the results of running a full minimization step with RAMBO_C after MIS-II. The last column shows that percentage of reduction in the literal count achieved by RAMBO_C for the MIS-II-optimized circuits. With this simple experiment a significant improvement is obtained

TABLE III
EXPERIMENTAL RESULTS FOR FINITE STATE MACHINES

Name	MISII+ Red. Removal			RAMBO_C				RAMBO_S			
	#F	#G	#C	#F	#G	#C	time	#F	#G	#C	time
cse	4	55	142	4	41	118	11.3	4	36	105	32.5
dk14	3	46	111	3	34	88	8.5	3	32	81	30.8
dk15	2	35	89	2	26	72	4.4	2	24	68	13.7
dk17	3	20	46	3	18	43	0.9	3	15	37	2.7
ex2	5	44	123	5	38	113	11.9	5	36	107	308.7
ex5	3	26	72	3	24	67	3.6	3	21	58	18.6
ex6	3	39	97	3	31	84	7.7	3	26	72	14.1
keyb	3	52	128	3	41	109	17.5	3	38	106	114.8
styr	4	88	237	4	63	196	67.2	4	63	194	306.7

TABLE IV
EXPERIMENTAL RESULTS ISCAS-89 EXAMPLES

Name	MISII+ Red. Removal			RAMBO_C				RAMBO_S			
	#F	#G	#C	#F	#G	#C	time	#F	#G	#C	time
s298	14	67	168	14	56	151	16.7	14	50	136	311.1
s344	15	83	210	15	77	199	56.4	15	73	189	774.1
s382	21	70	189	21	60	172	24.2	21	57	165	778.8
s444	21	87	225	21	68	192	39.2	21	63	175	913.5
s526(*)	21	105	274	21	80	220	4.4	21	70	193	14.4
s1196(*)	18	299	784	18	239	640	99.5	18	235	637	114.3
s1423(*)	74	403	984	74	361	909	45.3	74	360	906	124.6
s5378(*)	131	630	1531	131	580	1427	135.8	104	540	1325	292.7
s9234(*)	134	569	1491	134	521	1393	135.8	125	520	1382	450.0
s13207(*)	149	251	912	149	226	863	22.6	124	220	815	391.1
s15850(*)	443	1834	4811	443	1686	4526	1596.3	437	1670	4486	10872.0

* Recursive learning is not used.

for most examples. When compared to the best results reported with MIS-II in [19], best results are obtained for *C3540*, *C432*, *C5315*, *C6288*, *C7552*, *apex6* and *rot*, even though the MIS-II results were obtained by running several variations of different scripts repeatedly. Best results are also obtained for *C1355* and *C499* with RAMBO_C alone.

Table III shows the results for several MCNC benchmark finite state machines. Table IV shows the results for some of the ISCAS89 benchmark circuits. The results in the column titled MIS-II+Red. Removal were obtained after the following two steps: 1) combinational optimization with MIS-II, using *script2* in [19]; 2) sequential redundancy removal using MIRACLE_S [9] (only for the large, very redundant examples) and the redundancy identification procedure described in Section III, with Recursive Learning [15] and a maximum recursion depth of 3. These results were used as the initial circuit for the experiments in the next two columns. Column titled RAMBO_C shows the results after several iterations of combinational optimization with RAMBO_C. Column titled RAMBO_S shows the results of several iterations of sequential optimization with RAMBO_S. The flip-flop count (#F), gate count (#G), connection count (#C), and CPU time consumed in SUN SPARC2 workstation are listed. Some of the results after redundancy removal are already remarkable (*s5378*, *s13207*). Flip-flop counts are reduced by RAMBO_S for some large examples, namely *s5378*, *s9234*, *s13207*, *s15850*, even though recursive learning was not used for these examples.

The computational time grows exponentially with the maximum recursion depth allowed for recursive learning. For the large ISCAS 89 examples, (those marked with *) we did not use recursive learning for redundancy addition and removal. Note that for these examples the CPU time ratio of RAMBO_S to RAMBO_C is considerably smaller

than the same ratio for the examples where recursive learning was used.

VIII. CONCLUSIONS AND FUTURE WORK

We proposed a method for optimizing multilevel combinational and sequential logic networks. The networks are optimized through iterative addition and removal of redundancies. Connections that can be added without changing the functionality of the network and will create new redundancies after their addition can be identified through the identification of mandatory assignments and their implications. The experimental results show that the best results are obtained for most of the ISCAS-85 combinational benchmark circuits. The proposed method is able to minimize synchronous sequential circuits without any restriction in their structure. However, the simplicity of the transformations used with the current greedy approach limits the chances of visiting many different state assignments. Future work is directed toward the finding of more powerful transformations that will provide the means to step out of local minima in the state assignment.

ACKNOWLEDGMENT

The authors would like to thank Dr. S. Mallela for first suggesting the use of absolute dominators for this work.

REFERENCES

- [1] R. K. Brayton, R. Rudell, A. Sangiovanni-Vincentelli, and A. R. Wang, "Multilevel interactive logic optimization system," *IEEE Trans. Computer-Aided Design*, vol. 6, no. 6, pp. 1062-1081, Nov. 1989.
- [2] K. A. Bartlett et al., "Multilevel logic minimization using implicit don't cares," *IEEE Trans. Computer-Aided Design*, vol. 7, no. 6, pp. 723-740, June 1988.

- [3] D. Bostick *et al.*, "The boulder optimal logic design system," in *Proc. Int. Conf. CAD*, Nov. 1987, pp. 62-65.
- [4] C. L. Berman and L. H. Trevillyan, "Global flow optimization in automatic logic design," *IEEE Trans. Computer-Aided Design*, vol. 10, pp. 557-564, May 1991.
- [5] G. Hachtel *et al.*, "Performance enhancements in BOLD using implications," in *Proc. Int. Conf. CAD*, Nov. 1988, pp. 94-97.
- [6] S. Muroga *et al.*, "The transduction method—Design of logic networks based on permissible functions," *IEEE Trans. Comput.*, vol. 38, no. 10, pp. 1404-1423, Oct. 1989.
- [7] S. Malik, E. M. Sentovich, R. K. Brayton, and A. Sangiovanni-Vincentelli, "Retiming and resynthesis: Optimizing sequential networks with combinational techniques," *IEEE Trans. Computer-Aided Design*, vol. 10, no. 1, pp. 74-84, Jan. 1991.
- [8] Y. Matsunaga and M. Fujita, "Multi-level logic optimization using binary decision diagrams," in *Proc. Int. Conf. CAD*, Nov. 1989, pp. 556-559.
- [9] K.-T. Cheng, "On removing redundancy in sequential circuits," in *Proc. 28th Design Automation Conf.*, pp. 164-169, June 1991.
- [10] H. Cho, G. D. Hachtel, and F. Somenzi, "Redundancy identification and removal based on implicit state enumeration," in *Int. Conf. Comput. Design (ICCD-91)*, Oct. 1991, pp. 77-80.
- [11] M. Schulz and E. Auth, "Advanced automatic test pattern generation and redundancy identification techniques," in *Proc. Fault Tolerant Computing Symp.*, June 1988, pp. 30-35.
- [12] Y. Matsunaga, M. Fujita, and T. Kakuda, "Multi-level logic minimization across latch boundaries," in *Proc. Int. Conf. CAD*, Nov. 1990, pp. 406-409.
- [13] H. Sato, Y. Yasue, Y. Matsunaga, and M. Fujita, "Boolean resubstitution with permissible functions and binary decision diagrams," in *Proc. 27th Design Automation Conf.*, June 1990, pp. 284-289.
- [14] K.-T. Cheng and Luis A. Entrena, "Multi-level logic optimization by redundancy addition and removal," in *European Conf. Design Automation (EDAC-93)*, Feb. 1993, pp. 373-377.
- [15] W. Kunz and D. K. Pradhan, "Recursive learning: An attractive alternative to the decision tree, test generation in digital circuits," in *Proc. Int. Test Conf.*, Oct. 1992, pp. 816-825.
- [16] T. Kirkland and M. R. Mercer, "A topological search algorithm for ATPG," in *Proc. 24th Design Automation Conf.*, June 1987, pp. 502-508.
- [17] P. Muth, "A nine-valued circuit model for test generation," *IEEE Trans. Comput.*, vol. C-25, pp. 630-636, June 1976.
- [18] J. P. Fishburn, "A depth-decreasing heuristic for combinational logic," in *Proc. 27th Design Automation Conf.*, June 1990, pp. 361-364.
- [19] H. Savoj, H.-Y. Wang, and R. K. Brayton, "Improved scripts in MIS-II for logic minimization of combinational circuits," in *Int. Workshop in Logic Synthesis*, Apr. 1991.