Retiming and Resynthesis: Optimizing Sequential Networks with Combinational Techniques

Sharad Malik, Ellen M. Sentovich, Robert K. Brayton, *Fellow, IEEE,* and Alberto Sangiovanni-Vincentelli, *Fellow, IEEE*

Abstract—**Sequential networks contain combinational logic blocks separated by registers. Application of combinational logic minimization techniques to the separate logic blocks results in improvement that is restricted by the placement of the registers; information about logical dependencies between blocks separated by registers is not utilized. Temporarily moving all the registers to the periphery of a network provides the combinational logic minimization tools with a global view of the logic. We propose a technique for optimizing a sequential network by moving the registers to the boundary of the network using an extension of retiming [8], [9], resynthesizing the combinational logic between the registers using existing logic minimization techniques, and replacing the registers throughout the network using retiming algorithms.**

I. Introduction

OVER THE past decade, combinational logic optimization has attained a significant level of maturity. The problems and approaches in combinational logic synthesis are well understood: almost fully for the two-level logic case (e.g., [2]), and to a lesser extent in the multilevel logic case (e.g., [1], [3]). In comparison, sequential synthesis is just beginning to be recognized as a problem domain in its own right. Most existing efforts in sequential synthesis can be classified into three categories. The first approach is to consider the portions of combinational logic between register boundaries and use combinational logic optimization techniques on these separate blocks. However, this is restrictive inasmuch as it does not permit the interactions between gates separated by register boundaries to be examined in the optimization process. The second approach ([8], [9]) involves moving registers across portions of combinational logic in order to minimize the cycle time or the number of registers used. This procedure, termed **retiming**, does not change any of the combinational logic blocks. Thus it does not consider further optimizations that could have been obtained with that option. The third approach considers sequential circuits as implementations of finite state machine (FSM) descriptions. Operations on state transition graphs (STG's) and results from automata theory have been used to optimize implementations of STG's. One drawback with this approach is that all manipulations and optimizations are attempted at the STG level and it is not clear how these are reflected in the final gate-level implementations of the machine. Researchers have proposed different cost criteria such as the number of edges and the number of states in the STG as metrics for operations at the STG level [6], [7]. Unfortunately, none of these is a consistent reflection of the gate-level complexity. A second limitation of this approach is that since it operates on the STG, it is necessary that the STG be available. This is not a problem if the circuit was synthesized from a high-level description and this description is retained. However, given a sequential circuit, extracting the STG is a formidable task (in the worst case it is exponential in the number of state bits, or equivalently, in the number of latches). The inability to extract the STG within a reasonable amount of computing time may make this approach inapplicable.

In this paper we describe a new approach towards optimizing sequential circuits. As in [8], [9] we assume a synchronous implementation with edge-triggered registers (equivalently known as D-flip-flops). We characterize the subnetwork in the sequential network for which the registers can effectively be ignored and the subnetwork be considered as a combinational block. This permits existing combinational logic optimization techniques to be used on this block. This approach is more powerful than the first of the approaches stated previously, since it examines interactions between portions of logic separated by registers. As a result, the optimization process makes full use of dependencies between gates. Converting this subnetwork to a combinational logic block can be viewed as a retiming process in which all the registers have been pushed to the periphery of the subnetwork. However, our technique is more powerful than conventional retiming in that we permit **negative registers** to be pushed to the periphery. This is equivalent to temporarily "borrowing" registers from the environment, and is a legitimate operation as long as these registers are "returned" to the environment at the end of the optimization process. This additional allowance is more powerful since it permits a larger portion of the logic to be viewed as a single block than was permitted by conventional register movements and retiming. Next, this combinational logic block may be resynthesized according to a specified cost function. This could be minimizing the area, the delay or meeting a particular area/delay tradeoff. Finally, the registers may be redistributed in this combinational block. We guarantee that there will be some legal redistribution of the registers even with the negative registers; i.e., we will be able to return the registers that were borrowed from the environment. The redistribution can be done while satisfying constraints such as minimizing the number of registers subject to a specified cycle time (if these constraints are satisfiable) by using the algorithms described in [8]. Since the optimization algorithms work directly on the gate-level netlist, they use the gate-level complexity as their cost function. As a result, the circuit quality can only improve, unlike algorithms that work on STG's.

The remainder of this paper is organized as follows. Section II gives the theoretical formulation and results on which our

Manuscript received January 1, 1990. This work was supported by the National Science Foundation under Grant EMC-8419744 and by DARPA under Grant N00039-C-87-0182. This paper was recommended by Guest Editor A. Sangiovanni-Vincentelli.

The authors are with the Department of Electrical Engineering and Computer Science, University of California, Berkeley, CA 94720.

IEEE Log Number 9039377.

0278-0070/91/0100-0074$01.00 © 1991 IEEE

approach is based. In Section III we consider the application of this formulation to the problem of optimizing sequential circuits. Next, in Section IV we present results obtained on several classes of sequential circuits. Finally, we conclude with a summary of the contributions made by this work.

II. Theoretical Formulation

We model a sequential circuit by a directed acyclic graph called a **communication graph**[1] where each vertex v represents either

a) an input/output pin,
b) a combinational logic block.

The vertices in the graph are connected by directed edges. We place the restriction that each input pin has no incoming edges and exactly one outgoing edge, and that an output pin has no outgoing edges and exactly one incoming edge. An **internal edge** connects vertex u to vertex v if both u and v represent combinational logic blocks, and the logic represented by v explicitly depends on the value computed at u. A **peripheral edge** connects either an input pin to the logic block that uses that input or connects a logic block that computes the value of an output to the corresponding output pin. Each edge e has a corresponding weight $w(e)$ representing the number of registers between the two vertices it connects. The registers are edge-triggered D-flip-flops. An example of a communication graph is shown in Fig. 1. We use the terms circuit, network, and graph interchangeably whenever there is no ambiguity.

A **path** between two vertices v_1 and v_2 in the graph is a sequence of edges from v_1 to v_2. The weight of a path is the sum of the weights of all the edges along the path. In Fig. 1, the path from input i_1 to output o_1 has weight 2, while the path from input i_2 to o_1 has weight 3.

2.1. Retiming: An Overview

Retiming is an operation on a communication graph whereby registers are moved across logic blocks in order to minimize the clock cycle or the number of registers while maintaining the behavior of the circuit. Retiming algorithms were first proposed by Leiserson *et al.* [8], [9]. The movement of registers can be quantified by an integer $L(v)$ (called the lag of v) for each vertex v, which represents the number of registers that are to be moved in the circuit from each out-edge of vertex v to each of its in-edges.

Definition 1: A **legal retiming** is the assignment of an integer $L(v)$ to each vertex in the communication graph such that a) $L(v) = 0$ if v is an I/O pin and b) $w(e) + L(v) - L(u) \geq 0$ where e is the edge from vertex u to vertex v.

The edge weights of the retimed circuit, $w_r(e) = w(e) + L(v) - L(u)$, must be non-negative for all edges e, representing a non-negative number of registers connecting the two logic blocks. A legal retiming has been proven [9] to generate a circuit that is functionally equivalent to the original circuit. The circuit shown in Fig. 1 can be retimed by assigning a lag of -1 to vertex c ($L(c) = -1$) and a lag of 0 to all other vertices. The resulting retimed circuit is shown in Fig. 2. Note that for any legal retiming the path weights from the inputs to the outputs are unchanged.

[1]This is related to the definition of a communication graph presented in [9].

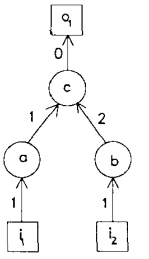


Fig. 1. Communication graph.

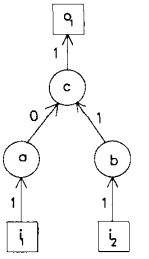


Fig. 2. Retimed circuit.

2.2. Extensions to Retiming

The retiming operation can be extended by introducing the concept of a "negative" register, that is, an edge weight in the graph that is negative. We permit negative edge weights on peripheral edges only. Allowing a negative edge weight n on a peripheral edge is equivalent to "borrowing" n registers from the environment. The registers may be "returned" by a subsequent retiming step whereby n registers are forced to each edge with weight $-n$. The observation that the peripheral edge weights can temporarily take on negative values allows retiming operations and subsequent optimizations that would otherwise not be possible. An example circuit is shown in Fig. 3(a) (in schematic drawings, combinational logic blocks are represented by conventional gate symbols or circles, and registers by rectangles). If a lag of -1 is assigned to the gate $g2$, the edge between input e and gate $g2$ would have weight -1, as Fig. 3(b). This is equivalent to borrowing a register at input e, which is indicated by the label -1 on the register at input e. During subsequent combinational resynthesis, the redundant connection from a to $g1$ allows the removal of gate $g1$ (Fig. 3(c)). Finally the circuit is retimed with $L(g2) = 1$. This returns the register borrowed at input e resulting in the circuit shown in Fig. 3(d). This smaller implementation could not be obtained without allowing the edge weight to temporarily take on a negative value.

We define in addition to legal retiming, a specific type of retiming that exploits the negative concept while pushing the registers to the boundaries of a network.

Definition 2: A *peripheral retiming* is a retiming such that a) $L(v) = 0$ where v is an I/O pin and b) $w(e) + L(v) - L(u) = 0$ where e is an internal edge from vertex u to vertex v.

A peripheral retiming moves all registers to the peripheral edges, leaving a purely combinational logic block between two sets of registers. For example, by assigning a lag of 1 to vertex

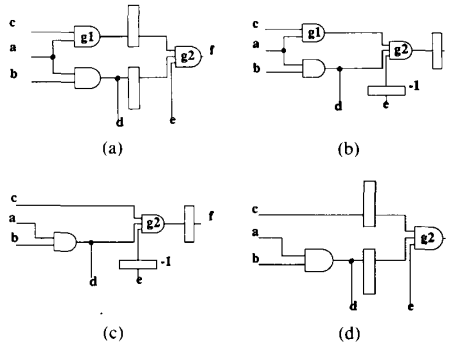


Fig. 3. Example: use of negative register. (a) Example circuit. (b) Borrow. (c) Resynthesize. (d) Return.

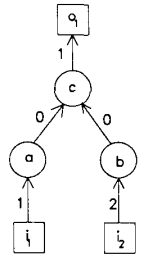


Fig. 4. Peripherally retimed circuit.

b in Fig. 2, we obtain the circuit in Fig. 4, which is a peripheral retiming of both the circuit in Fig. 1 and that in Fig. 2.

Note that the definition of a peripheral retiming permits negative edge weights on the peripheral edges, which corresponds to the negative register concept presented at the beginning of this section. Permitting negative registers on peripheral edges is a legitimate operation as shown by the following theorem.

Theorem 1: A circuit that undergoes a peripheral retiming and a subsequent legal retiming is equivalent to the original circuit.

Proof: See the Appendix. □

2.3. Conditions for Peripheral Retiming

Not all circuit structures permit a peripheral retiming: the circuit in Fig. 5 has no peripheral retiming because the register cannot be pushed to the output or to the input without leaving one of the internal edges with a nonzero weight. For example, if the register is moved toward output o_2, a weight of -1 is forced on the edge between vertices *a* and *d*; if this negative weight is pushed toward the inputs, a weight of 1 remains on the edge between vertices *a* and *c*. Regardless of how the register is moved, a nonzero weight will result on one of the internal edges between vertices *a* or *b* and *c* or *d*.

It is important to characterize circuit structures that allow peripheral retiming since for these circuits we can apply combinational optimization techniques on their entire combinational logic block. For this purpose we define the **path weight matrix** of a network.

Definition 3: A **path weight matrix, W**, of a sequential network is an $m \times n$ matrix, where

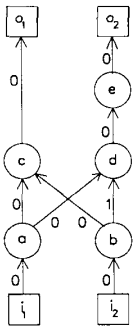


Fig. 5. Circuit with no valid peripheral retiming.

1) m is the number of inputs;
2) n is the number of outputs;
3) $W_{ij} = *$ if no path exists between input i and output j;
4) $W_{ij} = \sim$ if two paths between input i and output j have different weights;
5) $W_{ij} = \Sigma_{\text{path } i_i \to o_j} w(e)$ if all paths between input i and output j have the same weight.

In addition, we define the satisfiability condition on the path weight matrix, which is intimately related to the existence of a peripheral retiming.

Definition 4: A matrix **W** is **satisfiable** if a) $W_{ij} \neq \sim$, $\forall_i$, $\forall_j$; and b) $\exists\alpha_i$, $\exists\beta_j$, $1 \le i \le m$, $1 \le j \le n$, $\alpha_i, \beta_j \in I$ such that for each $W_{ij} \neq *$, $W_{ij} = \alpha_i + \beta_j$.

Since the communication graph is acyclic, only acyclic circuits can have satisfiable path weight matrices. Optimization of sequential circuits with a cyclic structure is described in Section III.

Finally, we state the relationship between a satisfiable path weight matrix and the existence of peripheral retiming.

Theorem 2: A sequential network has a peripheral retiming if and only if its path weight matrix is satisfiable.

Proof: See the Appendix. □

Note the signficance of this result: it gives a complete characterization of the class of sequential circuits for which all the registers can be pushed to the periphery allowing resynthesis on the combinational block.

A peripheral retiming involves finding a set of $\alpha's$ and $\beta's$ that satisfy the path weight matrix, and moving the registers accordingly. The path weight matrix contains information about the number of registers between each input and each output. α_i and β_j are the number of registers that appear at the ith input and the jth output edge, respectively, in the peripherally retimed circuit. A matrix that is satisfiable has no $\sim$ entries, and has at least one set of $\alpha_i s$ and $\beta_j s$ such that $\alpha_i + \beta_j = W_{ij}$. For the circuit in Fig. 1, the path weight matrix is as follows:

	o_1
i_1	2
i_2	3

and can be satisfied by choosing, for example, $\alpha_1 = 1$, $\alpha_2 = 2$, $\beta_1 = 1$, resulting in the circuit shown in Fig. 4.

Note that the path weight matrix for the circuit in Fig. 5, which had no peripheral retiming, is as follows:

	o_1	o_2
i_1	0	0
i_2	0	1

and it is easily checked that no α_i, β_j exist by applying the conditions necessary to satisfy the matrix. This yields

$$\alpha_1 + \beta_1 = 0 \quad (1)$$

$$\alpha_1 + \beta_2 = 0 \quad (2)$$

$$\alpha_2 + \beta_1 = 0 \quad (3)$$

$$\alpha_2 + \beta_2 = 1. \quad (4)$$

Subtracting (1) from (2):

$$\beta_2 - \beta_1 = 0.$$

Subtracting (3) from (4) yields

$$\beta_2 - \beta_1 = 1.$$

The contradiction implies that the path weight matrix is not satisfiable.

Computing the path weight matrix and finding a satisfying assignment of $\alpha_i's$ and $\beta_i's$ for that matrix can be done in time $O(e \cdot \min(n, m))$, where e is the number of edges, and n, m, the number of inputs and outputs, respectively, in the communication graph. The path weight matrix is computed by doing a depth-first traversal of the graph. At each node, an array of integers is stored representing the number of latches between that node and each of the primary outputs. When a node is reached during the traversal, the array entries for that node are computed by simply duplicating the entries from its fan-out nodes. If there is an entry that differs between two fan-out nodes, the corresponding entry for the current node is $\sim$; this will result in a matrix that cannot be satisfied and the computation is aborted.

Note that a path weight matrix will be either unsatisfiable or have an infinte number of solutions (e.g., given a particular solution, another valid solution can be obtained by adding integer j to the $\alpha's$ and subtracting j from the $\beta's$). Any solution can be used to obtain a peripheral retiming. For simplicity, we choose $\alpha_1 = 0$. This choice forces $\beta_1 = W_{1i}$, which in turn forces $\alpha_i = W_{i1} - \beta_1$. Each entry in the matrix is then checked to ensure that $\alpha_1 + \beta_j = W_{ij}$; if a violation occurs, the matrix is not satisfiable and no peripheral retiming exists for the circuit. The arbitrary selection $\alpha_1 = 0$ may not be the only assignment necessary to compute a complete set of $\alpha's$ and $\beta's$. The circuit may have subcircuits which are disjoint, leading to a corresponding disjoint matrix with many * entries. In this case, an arbitrary assignment to an α or a β must be made for each disjoint submatrix.

2.4. *Legal Resynthesis Operations*

Permitting negative registers on the peripheral edges is a legitimate operation as long as the resynthesized circuit has a legal retiming. This leads us to ask the following question: can we guarantee that the resynthesized circuit always has a legal retiming? To examine this further we need to define a synchronous communication graph[2].

[2]This is related to the definition of a synchronous circuit presented in [9].

Definition 5: A synchronous communication graph is one in which each path between an input pin and an output pin has a non-negative path weight.

The following theorem precisely states the conditions under which a legal retiming exists.

Theorem 3: A communication graph has a legal retiming if and only if it is synchronous.

Proof: See the Appendix. □

Note that since the initial communication graph had no negative edges (it represents a real circuit), it is synchronous. Peripheral retiming preserves the synchronous property since retiming does not change the path weight between an input and an output pin. However, resynthesis can change the communication graph, and hence, it may destroy the synchronous property.

Let us see how this can happen. Let G_1 be the communication graph before resynthesis and G_2 be the graph after resynthesis. If there was a path between input i and output j in G_1 and there is a path between them in G_2, then the path weight for this path in G_2 is $\alpha_i + \beta_j$. This is the same as the path weight W_{ij} in G_1. Since G_1 was synchronous, this path weight is non-negative. Now consider the case in which no path existed in G_1 between input i and output j and resynthesis creates a path. The path weight for this path in G_2 is $\alpha_i + \beta_j$. Since α_i and β_j may be negative and G_1 did not force a non-negativity constraint on $\alpha_i + \beta_j$ (since no path existed between input i and output j), it is possible that $\alpha_i + \beta_j$ may be negative, thus destroying the synchronous property. Note that output j does not actually depend on input i; however, resynthesis created a *pseudodependency* between the two.

An example is shown in Fig. 6(a). This circuit has a peripheral retiming shown in Fig. 6(b). Resynthesis discovers that the three-input OR gate g_1, can be replaced by a two-input OR gate g_2 (Fig. 6(c)). The communication graph for this circuit is not synchronous since there exists a path of negative weight (-1) between input a and output *out*1. By Theorem 3 we know that this circuit has no legal retiming.

Thus resynthesis must ensure that it does not introduce a pseudodependency with a negative path weight; this is the only condition that the resynthesis must satisfy. This condition can be checked easily after resynthesis and the resynthesis rejected if this does happen.

III. Optimizing Sequential Circuits

We now focus our attention on applying the techniques discussed in this paper to general sequential circuits. For those circuits that can be peripherally retimed, the entire interior logic block can be optimized and the registers replaced in the circuit. In this section, we concentrate on those that cannot initially be peripherally retimed.

As is illustrated by the circuit in Fig. 7(a), an acyclic circuit may not have a satisfiable path weight matrix, and thus no peripheral retiming exists. In this case, satisfiable subcircuits (subcircuits whose path weight matrices are satisfiable) are identified and created by breaking the appropriate nets. Each subcircuit is optimized separately, and the subcircuits are then reconnected. Consider the circuit in Fig. 7(a). Breaking net ***x*** yields the subcircuit shown in Fig. 7(b). ***x*_out** represents additional outputs of the subcircuit, and ***x*_in** represents additional inputs. This subcircuit is satisfiable and the results of Section

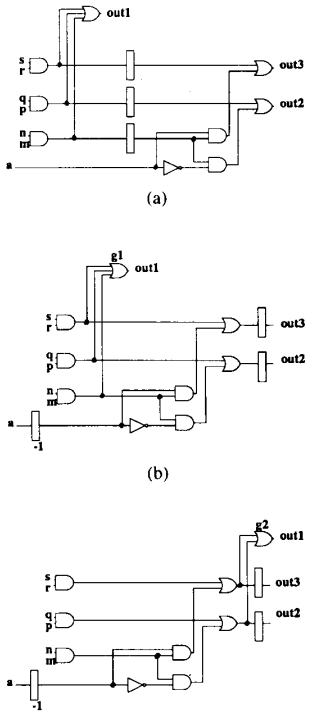


Fig. 6. Introducing pseudodependence with negative path weight.

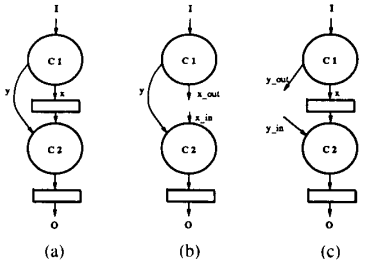


Fig. 7. Acyclic circuit with no peripheral retiming.

II can be directly applied. Finally, a circuit equivalent to the original circuit can be optimized by reconnecting the net x. Alternately, net y can be broken and the corresponding subcircuit, Fig. 7(c) similarly optimized. Note that the optimization of subcircuits (b) and (c) can lead to very different results.

In the case of sequential circuits that have cycles in them, we first need to make them acyclic. Therefore, the first step is to choose a set of nets to cut such that all cycles are broken. However, this may not be sufficient: the resulting acyclic circuit may still have a path weight matrix that is not satisfiable. For example, breaking net z of the circuit in Fig. 8 will break the cycle, but as in Fig. 7(a), net x or net y still must be broken to make the path weight matrix satisfiable.

In most cases, there will be several choices of where to make cuts in the logic to create a satisfiable path weight matrix. While it is not known *a priori* which cut will yield the best results after optimization, it is simple enough to provide an interactive

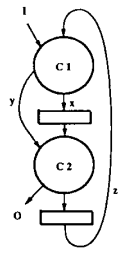


Fig. 8. Cyclic circuit.

environment for the designer to experiment with several different cuts. (We have provided such an environment as part of SIS, a sequential interactive system built on top of MISII.)

We now look at an example of a sequential circuit that has a cyclic structure. Fig. 9(a) shows a gate-level schematic of a FSM implementation. This circuit is optimal with respect to conventional logic minimization of the combinational logic between the registers. There are no redundant gates or connections. We break the cycles by cutting the nets $p1$ and $p2$. This results in pseudo-inputs $p1\_in$ and $p2\_in$ and pseudo-outputs $p1\_out$ and $p2\_out$ in the circuit. The circuit is then redrawn with the signal flow unidirectional (Fig. 9(b)). A peripheral retiming of this circuit is shown in Fig. 9(c). An optimization of the combinational block simplifies the logic part by observing that the output of the NOR gate may be replaced by the constant value 1 without changing the functionality of the circuit. This simplified circuit is shown in Fig. 9(d). The circuit is retimed with a legal retiming (Fig. 9(e)). The feedback connections are made and the final circuit is shown in Fig. 9(f). This circuit has three fewer gates than the initial circuit; this represents a significant gain.

3.1. Summary of the Algorithm

Given any sequential circuit, we can optimize it by identifying and creating subcircuits whose path matrices are satisfiable, pushing the registers to the boundaries of the subcircuits, resynthesizing the logic blocks, applying a legal retiming, and reconnecting any nets that may have been broken to create the subcircuits.

The algorithm is summarized as follows.

1) Choose a set of nets to cut such that all cycles are broken.

2) Formulate the path weight matrix for the circuit. If necessary, identify satisfiable subcircuits and cut additional nets to create these subcircuits.

3) Compute α_i and β_j for $1 \le i \le m$, $1 \le j \le n$.

4) Place α_i registers after each input i and β_j registers before each output j; remove (replace by wires) all other registers.

5) Resynthesize the interior combinational logic block using any valid technique.

6) Formulate the path weight matrix for the retimed circuit.

7) If the path weight matrix has no negative entries, find a legal retiming for the circuit according to a cost criterion (minimize clock cycle, minimize state), else reject the resynthesis and go to step 5.

8) Reconnect the subcircuits.

3.2. Relationship to State Assignment

It is of interest to determine the relationship of these optimization techniques to state assignment for FSM's. We were

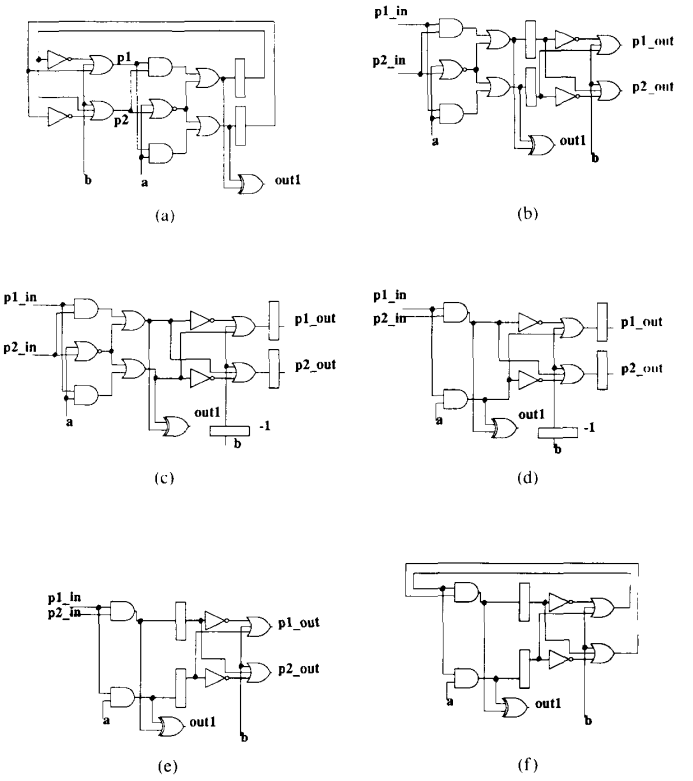


Fig. 9. Example FSM optimization.

interested in examining the following questions for a given FSM: given a circuit implementation with some state assignment, is it possible to obtain any equivalent implementation with any other state assignment using only retiming and resynthesis? We were able to prove the following result in this direction.

Theorem 4: Given a machine implementation M_1 corresponding to a state transition graph G, with a state assignment S_1, it is always possible to derive a machine M_2 corresponding to the same state transition graph G, and a state assignment S_2 by applying only a series of resynthesis and retiming operations on M_1.

Proof: See the Appendix. □

The above result does not include the possibility of modifying the STG. However, this may be only a limitation of our proof technique rather than that of the optimization technique.

3.3 Computing Equivalent States Across Optimizations

The migration of registers raises concern about the starting state of FSM's in the minds of circuit designers and testers. The question that needs to be answered here is as follows: given the starting state of the initial circuit, how do we determine the starting state of the circuit obtained by applying retiming and resynthesis techniques on the original circuit? This is contained in the more general problem of determining a state in the final circuit that is equivalent to a known state in the initial circuit. In [16] a procedure is provided that handles this problem for retimed circuits. Since combinational resynthesis does not migrate any registers or change the function of the input gate of a register, there is no change in the state information in this step. Thus using the procedure outlined in [16] at each retiming step is sufficient to tackle this problem. Related to this is the issue of initialization sequences. Typically, the design of a state machine is accompanied by the determination of an initialization sequence, i.e., a sequence of input vectors that is guaranteed to bring the machine to some known state (referred to as the starting state) independent of the state it is currently in. Thus the machine may start in any possible state when it is powered on and going through the initialization sequence brings it to the known starting state. The initialization sequence may be as simple as a single input on a reset line. Reference [16] describes how the initialization sequence for a retimed circuit is determined given the initialization sequence for the initial circuit. By an argument similar to that given for determining equivalent states, this procedure is applicable when both retiming and resynthesis are used.

We would like to point out that in general it is possible that we may be able to find a state equivalent to the starting state in a retimed circuit. After retiming, some or all of the registers could possibly have been replaced with wires. The only values that these wires can have are those that are consistent with the logical structure of the combinational network (i.e., no value that is part of the satisfiability don't care set for the combinational network). If the initial state is inconsistent with this then it will not be possible to find an equivalent state in the retimed circuit. For example, let us consider a network which has an AND gate whose output signal feeds two registers $R1$ and $R2$. Also, the outputs of these registers are primary outputs of this

network. Consider the state ($R1 = 1, R2 = 0$). If we retime this circuit to move registers to the inputs of this gate then we can never obtain a state in this new circuit that is equivalent to ($R1 = 1, R2 = 0$) in the original circuit. Note that we never specified how the state ($R1 = 1, R2 = 0$) was reached for this circuit. The only way it could have happened was if there was some implicit reset circuitry. However, if all the logic associated with the reset circuitry is made explicit then it can be shown that this problem can never occur. The overhead associated with making this circuitry explicit is only needed to guarantee that we will always be able to find an equivalent state after retiming. Thus the reset circuitry needs to be put in only after the algorithm given in [16] determines that no equivalent state exists in the retimed circuit.

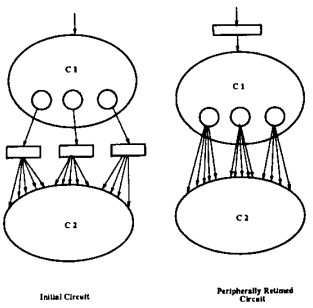


Fig. 10. Register outputs form a high fan-out cutset.

IV. Experimental Results

In this section we describe the results of an implementation of the ideas presented in this paper towards area optimization of sequential circuits. (These ideas have been used successfully for performance optimization; these results have been reported in a separate paper [11].) Unlike combinational logic circuits, there are no accepted sequential circuit benchmarks that are used to evaluate logic optimization algorithms. Therefore, our choices were limited in selecting the examples for our experiments. The examples we used came from the three different sources; we describe separately our experiences with each of these.

4.1. MCNC FSM Benchmarks

The first class of sequential circuits that we looked at were circuits generated from the MCNC FSM benchmarks [10]. These benchmarks are state transition tables. Circuit implementations for these were obtained by the following procedure. First state assignment was done using the program NOVA [17] to give a two-level circuit. This was first minimized by a two-level minimizer, ESPRESSO [2] and then followed by a multilevel minimization using MISII [3]. An interactive environment was provided that enabled the generation of acyclic satisfiable circuits. Retiming and resynthesis techniques were used in several of these for each example. Surprisingly, for all the circuits there was either no area improvement or too little to be of any significance. It is instructive to examine why this is so. The following observation enables us to understand this better. The circuits for the FSM's were constructed from two-level descriptions. It is common in two-level circuits for the primary inputs to fan out to a large number of gates (each primary input variable is used in a large number of product terms). Even when multilevel optimization is used on the two-level circuits the high fan-out property of the primary inputs is retained. Since register outputs are primary inputs to the combinational logic, they share this high fan-out property. For the benchmark FSM circuits, the average number of fan outs for register outputs is 9 as compared to 2 for the internal nodes. When the registers are moved to the periphery, their input nodes have high fan out (see Fig. 10). We now examine the implications of this on the optimization techniques included in MISII.

MISII has two main phases of optimizations: circuit restructuring and node simplification. We shall look at both of these separately.

The circuit restructuring phase of MISII operates by collapsing low fan-out nodes into their fan-out nodes and then restructuring these large resulting nodes using the algebraic techniques of cube and kernel extraction. Note that since the register input nodes have high fan out in the peripherally retimed circuits, these nodes form a high fan-out cutset of this network. This effectively restricts the restructuring to each side of the original register boundaries. However, this restructuring had already been exploited, and therefore, nothing is gained by migrating the registers. Completely collapsing the circuit, and thereby removing the barrier to restructuring, is computationally too expensive.

The node simplification phase of MISII constructs the satisfiability don't care set and simplifies each node using two-level minimization with this don't care set. Typically this don't care set is large and a filter is used that extracts only part of this don't care set for a node [13]. This is determined by looking at the topology of the network. For the topology that we are working with, a network where the original register inputs form a cutset, the filtering would restrict the don't cares for a node to be generated only from other nodes that are on the same side of the cutset. Again this has already been exploited and nothing new is obtained from migrating the registers. We then disabled the filter and used the complete satisfiability don't care set. However, this did not improve any of the results. This was partially to be expected since the filter has been designed so that the quality of the results are almost as good as what can be obtained with the entire satisfiability don't care set. Finally, we included node simplification using a subset of the observability don't cares for a node [14]. We believe these don't cares are very useful in this case since the observability of the nodes in subcircuit C_1(see Fig. 10) is changed by adding subcircuit C_2 and thus additional simplification is possible.[3] However, the observed improvement was negligible. There are two possible reasons for this. First, only a subset of the observability don't cares are being used. This may not be sufficient. Second, the depth of the sequential part of these circuits is small (an average of 5 for the benchmark FSM circuits). It has been observed that observability don't cares are more useful in deeper circuits as compared to shallower ones. This may explain why they do not contribute significantly to the simplification process in this case.

In conclusion, for the FSM circuits, the lack of improvement may be explained by the limitations of the combinational optimization techniques used. By considering logical relationships across latch boundaries we are giving the combinational optimization tools additional information that they can exploit.

[3]The nodes in C_1 were initially observable at the latches (for the combinational part). These observation points have been removed now.

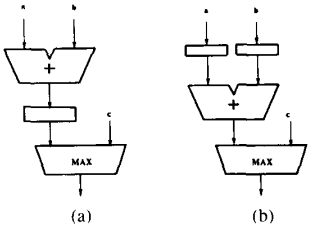
However, the existing tools are not powerful enough to exploit this additional information. Current research in combinational logic optimization techniques (e.g., [12]) holds some promise in terms of discovering more powerful techniques that do circuit restructuring without collapsing and algebraic factoring.

4.2. Pipelined Circuits

The next set of circuits we examined were pipelined data paths obtained from a speech recognition chip [15]. These gave us some insight as to when there is inherently no potential for further improvement and thus no use in expending any further effort. Fig. 11(a) shows one of these. Here the function MAX($a + b$, c) is performed over two cycles, with the addition being done in the first stage and the selection of the maximum done in the second stage. We did not obtain any area improvement for any of these circuits. Let us examine the reasons for this for the circuit in Fig. 11(a). Fig. 11(b) shows the same circuit with a peripheral retiming. Note that if $c = 0$, then the output of this circuit is $a + b$. Thus even though the output of the adder is not *explicitly* observable for this circuit, it is *implicitly* observable since it can be passed on to the output of the circuit by setting c to be 0. Thus no additional observability don't cares are generated by the cascade of the two logic blocks. We also note that the implicit observability of the adder forces the adder IO-map to be a Boolean function as opposed to a Boolean relation [4], i.e., no two outputs of the adder are equivalent as far as the MAX logic block is concerned. Thus there is no flexibility in changing the logic function of the adder block. It has been our experience that implicit observability is a general characteristic of data path circuits and this property does not make them amenable to further area optimization using retiming and resynthesis techniques.

Finally, we experimented with some circuits that we generated ourselves. In each case, we designed a two-stage pipeline with a register bank separating two combinational networks (C_1 and C_2). The first of these (adder_comp) is a circuit similar to one introduced in [4]. Here C_1 is an adder and C_2 compares the result of the first stage with a constant value to give a 1-b result. The other circuits have been generated by selecting C_1 and C_2 from the MCNC combinational benchmark set. The results for these circuits are shown in Table I. We observe that in each of these cases the ability to simplify the two circuits together results in significant reduction of logic. (In each case MISII with observability don't care simplification was used both on the separate networks and for the single network generated after the obvious peripheral retiming.) For example, if we consider adder_comp we see that the final circuit has only half the number of literals as the initial circuit and one register compared to five in the initial circuit. The version of adder_comp described in this table is a 4-b implementation. The combinational circuit generated by peripheral retiming was small enough that MISII was able to collapse the circuit and simplify it. When the same experiment was run with an 8-b version, no improvement was seen even though we could design a functionally equivalent circuit of half the size. While MISII was able to see the logical dependencies in the smaller 4-b version, it was not powerful enough to exploit the additional information given to it in the larger 8-b version. This result gives further support to the conclusions of Section IV-4.1.

It has been suggested that hardware generated by high-level synthesis systems might have a suboptimal initial register placement, and these techniques would be useful in that environment [5] even with existing combinational tools. This bears further investigation.



(a) (b)

Fig. 11. Example from a data path.

TABLE I
RESULTS: SELF-DESIGNED CIRCUITS

Example	Initial Circuit C_1 Size	C_2 Size	$C_1 + C_2$ Size	Regs.	Final Circuit Size	Regs.
adder_comp	43	5	48	5	24	1
misex1_con1	51	19	70	7	37	2
misex1_5xp1	51	113	164	7	122	10
vg2_rd4	86	125	211	8	178	4

Size: Total number of factored-form literals for the circuit. Regs: Number of registers in the circuit.

V. CONCLUSIONS

We have presented an approach towards optimizing sequential circuits by considering the subcircuits for which all registers can be effectively ignored. This permits existing combinational techniques to be used on these subcircuits. We have presented a complete characterization of these subcircuits in as much as these are maximal, i.e., no larger subcircuits exists, containing these, that can be simplified using combinational techniques. In addition we guarantee a legal retiming at the end of combination resynthesis. The concept of "borrowing" latches from the environment is an important one, for it extends the class of circuits that we can optimize by this technique.

Surprisingly, the application of these techniques for area optimization to some example circuits did not yield any significant gain. However, further analysis exposed two facts: 1) existing combinational optimization techniques have some limitations and are not powerful enough to exploit the potential of this new approach in sequential design (better results would be obtained with a combinational optimization program that exploits observability don't care conditions, and that takes a more global view of the logic while restructuring the circuit), and 2) some circuits (pipelined data paths) have inherently no potential for further optimization using these techniques. Our current experiences are limited to the circuits that we had access to and the combinational logic optimization techniques available to us. No generalizations can be made about the utility of these techniques for other circuits, or with other combinational resynthesis methods (including human designers). The theoretical framework for register migration that we have provided is complete and can be used with any arbitrary combinational optimization techniques and circuits. In this paper we have considered only area optimization in the resynthesis phase. The application of these ideas towards performance optimization of sequential circuits has been reported separately [11].

APPENDIX

Theorem 1: A circuit that undergoes a peripheral retiming and a subsequent legal retiming is equivalent to the original circuit.

Proof: Let C_1 be the original circuit and C_2 be the peripherally retimed circuit obtained with retiming r. Let α_i and β_j be the number of registers at the ith input and jth output pin in C_2. Let C_3 be the circuit obtained after combinational resynthesis, $\mathcal{R}$, on the interior combinational logic and let C_4 be the circuit obtained after a legal retiming l on C_3.

Let $\alpha_{\min} = |\min(0, \alpha_i)|$ over all α_i and let $\beta_{\min} = |\min(0, \beta_j)|$ over all β_j. Consider the circuit C_5 obtained from C_1 by adding $\alpha_{\min}$ registers at each input pin and $\beta_{\min}$ registers at each output pin. $C_5 = \text{delay}(C_1, \alpha_{\min} + \beta_{\min})$, i.e., given an input-output vector sequence $(\mathcal{I}, \mathcal{O})$ for C_1, the input sequence $\mathcal{I}$ results in the output sequence $\mathcal{O}$ delayed by $\alpha_{\min} + \beta_{\min}$ cycles in C_5. Let C_6 be the circuit obtained by retiming C_5 with r. This is a peripheral retiming of C_5 with $\alpha_i' = \alpha_i + \alpha_{\min}$ and $\beta_j' = \beta_j + \beta_{\min}$. Note that this is a legal retiming since there are no negative registers, as α_i' and β_j' are non-negative and there are no other registers in the circuit. Here we will take recourse to the results in [9] that show functional equivalences with legal retimings to claim that C_6 is equivalent to C_5.[4] Combinational resynthesis, $\mathcal{R}$, of the interior combinational logic of C_6, resulting in the circuit C_7, obviously does not change its functionality since none of the functions of any primary outputs or register inputs are changed by this. Now, retiming l is applied to C_7 to result in C_8. Note that since l was a legal retiming for C_3 it must result in at least $\alpha_{\min}$ registers at each input pin and $\beta_{\min}$ registers at each output pin. Also, by transitivity, C_8 is equivalent to C_5. Hence, $C_8 = \text{delay}(C_1, \alpha_{\min} + \beta_{\min})$. Let C_9 be obtained from C_8 by removing $\alpha_{\min}$ registers from each input pin and $\beta_{\min}$ registers from each output pin. Thus $C_9 = \text{delay}(C_1, 0)$, i.e., C_9 is equivalent to C_1. Note that C_9 is identical to C_4 because the same resynthesis $\mathcal{R}$ and final retiming l were applied in order to obtain them, ensuring that they have the same gate and register netlists. Thus C_4 is equivalent to C_1. □

Theorem 2: A sequential network has a peripheral retiming if and only if its path weight matrix is satisfiable.

Proof: If Part: Let the α_i's and β_j's be the integers that satisfy the path weight matrix. The following lemma is used to aid the proof.

Lemma 1: For a non-I/O vertex v in the communication graph, let $f_i(v) = \alpha_i - \Sigma_{\text{path}\, i_i \to v} w(e)$, where i_i is some ith input pin and there is path from this pin to v. $f_i(v)$ is independent of i.

Proof: Let i_1 and i_2 be two inputs each with a path to vertex v. Then:

$$f_1(v) = \alpha_1 - \sum_{\text{path}\, i_1 \to v} w(e)$$

$$f_2(v) = \alpha_2 - \sum_{\text{path}\, i_2 \to v} w(e).$$

Let v have a path to some output o_1 (it must have a path to some output pin). The weight along the path from v to o_1 is

$$\sum_{\text{path}\, v \to o_1} w(e).$$

[4]Functional equivalence here is subject to being able to get the two circuits in equivalent states. The problem of finding equivalent states for the original and retimed circuits has been looked at in [16] and is discussed in Section III.

The path from i_1 to o_1 has weight

$$\sum_{\text{path}\, i_1 \to v} w(e) + \sum_{\text{path}\, v \to o_1} w(e) = \alpha_1 + \beta_1. \tag{5}$$

The path from i_2 to o_1 has weight

$$\sum_{\text{path}\, i_2 \to v} w(e) + \sum_{\text{path}\, v \to o_1} = \alpha_2 + \beta_1. \tag{6}$$

Subtracting (5) from (6) yields

$$\sum_{\text{path}\, i_2 \to v} w(e) - \sum_{\text{path}\, i_1 \to v} w(e) = \alpha_2 - \alpha_1.$$

Rearranging, we obtain

$$\alpha_1 - \sum_{\text{path}\, i_1 \to v} w(e) = \alpha_2 - \sum_{\text{path}\, i_2 \to v} w(e)$$

$$f_1(v) = f_2(v). \qquad \square$$

Using the result of the previous lemma, the following lemma completes the proof for this part.

Lemma 2: The following lag function, $L_p(v)$, results in a peripheral retiming: a) $L_p(v) = \alpha_i - \Sigma_{\text{path}\, i_i \to v} w(e)$ for each internal vertex; and b) $L_p(v) = 0$ for each I/O pin where α_i is the α associated with the input i which has a path to vertex v, and $\Sigma_{\text{path}\, i_i \to v} w(e)$ is the weight of any path from input i to v.

Proof: A peripheral retiming requires $L(v) = 0$ for each I/O pin, which is satisfied by the above definition of $L_p(v)$. In addition, $w(e)$ must be 0 for each internal edge e. Each edge weight in the retimed circuit can be expressed as follows:

$$w_r(e_{uv}) = w(e_{uv}) + L(v) - L(u)$$

where $w(e_{uv})$ is the weight of the edge from u to v. $L(u)$ can be expressed in terms of any α_i such that there is a path from the ith input to vertex u. Any such i suffices because $L(u)$ is independent of i by Lemma 1.

$$L(u) = \alpha_i - \sum_{\text{path}\, i_i \to u} w(e).$$

If a path exists from input i to vertex u, then there is a path from input i to vertex v, and $L(v)$ can be expressed in terms of input i:

$$L(v) = \alpha_i - \sum_{\text{path}\, i_i \to v} w(e).$$

Hence,

$$w_r(e_{uv}) = w(e_{uv}) + \alpha_i - \sum_{\text{path}\, i_i \to v} w(e) - \left(\alpha_i - \sum_{\text{path}\, i_i \to u} w(e)\right)$$

$$= w(e_{uv}) - w(e_{uv}) = 0.$$

For an input edge, vertex u is an input pin, so $L(u) = 0$. In addition, each input edge has the property $w(e_{uv}) = \Sigma_{\text{path}\, i_i \to v} w(e)$, yielding

$$w_r(e_{uv}) = w(e_{uv}) + \alpha_i - \sum_{\text{path}\, i_i \to v} w(e)$$

$$w_r(e_{uv}) = \alpha_i.$$

Similarly, for the output edges of the network, v is an output pin, so $L(v) = 0$. Thus:

$$w_r(e_{uv}) = w(e_{uv}) + L(v) - L(u)$$

$$= w(e_{uv}) - \left(\alpha_i - \sum_{\text{path}\, i_i \to u} w(e)\right).$$

For each output edge e between vertex u and I/O pin v,

$$\sum_{\text{path } i_i \to u} w(e) = \alpha_i + \beta_j - w(e_{uv})$$

so the weight of the edge in the retimed circuit can be expressed as follows:

$$w_r(e_{uv}) = w(e_{uv}) - \left[\alpha_i - (\alpha_i + \beta_j - w(e_{uv}))\right]$$

$$w_r(e_{uv}) = \beta_j.$$

Thus the specified lag function results in the desired edge weights for the internal and peripheral edges that are required by the peripheral retiming. □

Only if part: It suffices to show that a circuit with a peripheral retiming has a satisfiable path weight matrix. A circuit with a peripheral retiming has an integral (possibly negative) number of registers, α_i, at the ith input and an integral (possibly negative) number of registers, β_j, at the jth output, and no registers on any of the internal edges. Regardless of the path chosen, the number of registers between the ith input and the jth output is $\alpha_i + \beta_j$ (if a path exists). This is the (i, j)th entry of the path weight matrix. Thus the path weight matrix is satisfied by these α's and β's. □

Theorem 3: A communication graph has a legal retiming if and only if it is synchronous.

Proof: If part: The following lemma serves as the proof for this part.

Lemma 3: The following lag function results in a legal retiming: a) $L_l(v) = sp(v)$ for each internal vertex. $sp(v)$ is the weight of the shortest path to the outputs, i.e., the path with the least weight between vertex v and an output pin. b) $L_l(v) = 0$ for each I/O pin.

Proof: The edge weight in a retimed circuit is given by

$$w_r(e_{uv}) = w(e_{uv}) + L(v) - L(u)$$

where edge e_{uv} is from vertex u to vertex v and L is the lag function.

Let us consider an internal edge and the lag function given in the statement of the lemma:

$$w_r(e_{uv}) = w(e_{uv}) + sp(v) - sp(u)$$

For $w_r(e_{uv})$ to be non-negative, $w(e_{uv}) + sp(v) - sp(u) \geq 0$ or $sp(u) \leq w(e_{uv}) + sp(v)$. This is obviously true since the weight of the shortest path from u to an output pin can be no more than $w(e_{uv}) + sp(v)$, or the shortest path would be the edge (u, v) followed by the shortest path from v to an output pin. Thus this retiming results in all internal edges having non-negative weights.

Now consider an output edge. For an output edge, $L_l(v) = 0$:

$$w_r(e_{uv}) = w(e_{uv}) + L_l(v) - L_l(u) = w(e_{uv}) + 0 - sp(u).$$

Now, $w(e_{uv}) - sp(u) \geq 0$ since $sp(u) \leq w(e_{uv})$. Thus $w_r(e_{uv}) \geq 0$.

Finally, let us consider an input edge. $L_l(u) = 0$. Thus

$$w_r(e_{uv}) = w(e_{uv}) + L_l(v) - 0 = w(e_{uv}) + sp(v).$$

Now, $w(e_{uv}) + sp(v)$ is the shortest path between the input pin u and the outputs. We know that all path weights between an input pin and an output pin are non-negative. Thus $w(e_{uv}) + sp(v)$ is non-negative and $w_r(e_{uv}) \geq 0$ for an input edge.

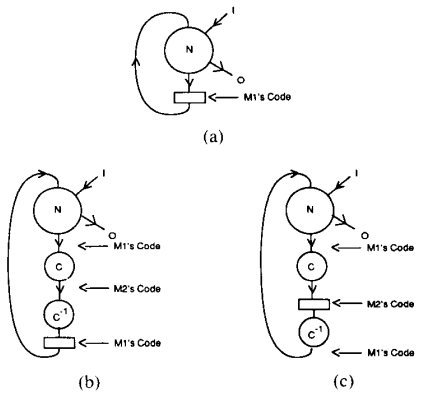


Fig. 12. Obtaining equivalent FSM implementations. (a) Machine M_1, (b) Resynthesize, (c) Retime to get M_2.

Thus the specified retiming is legal since all resulting edge weights are non-negative. □

Only if Part: If the resulting retiming is legal, then each edge weight in the retimed graph is non-negative. Thus the path weight between an input pin and an output pin must be non-negative. Retiming cannot change the path weight between an input pin and an output pin. Thus the path weight between an input pin and an output pin must have been non-negative before the retiming. Therefore, the communication graph was (and is) synchronous. □

Theorem 4: Given a machine implementation M_1 with a STG G, with a state assignment S_1, it is always possible to derive a machine M_2 with the same STG G, and a state assignment S_2 by applying only a series of resynthesis and retiming operations on M_1.

Proof: Given M_1 we would like to obtain M_2 using only a series of resynthesis and retiming steps. Fig. 12(a) shows the schematic for M_1. Since there is a one to one mapping between the states of M_1 and M_2, it is possible to construct a circuit C such that given the code for a state of M_1 as input, the output is the code for the corresponding state in M_2. Similarly, the inverse circuit C^{-1} can be constructed that takes a state of M_2 as input, and outputs the code for the corresponding state in M_1. Note that C followed by C^{-1} is the identity circuit, i.e., for this circuit, the output is the same as the input. This construction is shown in Fig. 12(b). The inputs to the state register are resynthesized as C followed by C^{-1}. Now, the state register may be moved to between C and C^{-1} by retiming as shown in Fig. 12(c). This circuit corresponds to the state assignments S_2. Any other circuit corresponding to state assignment S_2 may be obtained by the resynthesis of interior combinational logic. □

REFERENCES

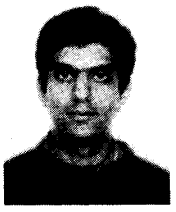
[1] D. Bostick, G. D. Hachtel, R. Jacoby, M. R. Lightner, P. Moceyunas, C. R. Morrison, and D. Ravenscroft, "The Boulder optimal logic design system," in *Proc. Int. Conf. on Computer-Aided Design*, 1987.

[2] R. K. Brayton, G. D. Hachtel, C. T. McMullen, and A. Sangiovanni-Vincentelli, *Logic Minimization Algorithms for VLSI Synthesis*. Hingham, MA.: Kluwer Academic, 1984.

[3] R. K. Brayton, R. Rudell, A. Sangiovanni-Vincentelli, and A. R. Wang, "MIS: A multiple-level logic optimization system," *IEEE Trans. Computer-Aided Design*, vol. CAD-6, pp. 1062–1081, Nov. 1987.

[4] R. K. Brayton and F. Somenzi, "Boolean relations and the incomplete specification of Boolean networks," in *Proc. VLSI 89 Int. Conf.*, 1989.
[5] G. DeMicheli, personal communication.
[6] S. Devadas, "Approaches to multi-level sequential synthesis," in *Proc. Design Automation Conf.*, 1989.
[7] S. Devadas and A. R. Newton, "Decomposition and factorization of sequential finite state machines," *IEEE Trans. Computer-Aided Design*, vol. 8, pp. 1206–1217, Nov. 1989.
[8] C. E. Leiserson, F. M. Rose, and J. B. Saxe, "Optimizing synchronous circuitry by retiming," in *Proc. Third Caltech Conf. on VLSI*, 1983.
[9] C. E. Leiserson and J. B. Saxe, "Optimizing synchronous systems," in *Proc. Twenty-Second Ann. Symp. on Foundations of Computer Science*, 1981, pp. 23–26.
[10] R. Lisanke, "Logic synthesis benchmark circuits," presented at Int. Workshop on Logic Synthesis, 1989.
[11] S. Malik, K. J. Singh, R. K. Brayton, and A. Sangiovanni-Vincentelli, "Performance optimization of pipelined circuits," in *Proc. Int. Workshop on Timing Issues in the Specification and Synthesis of Digital Systems*, 1990.
[12] S. Muroga, Y. Kambayashi, H. C. Lai, and J. N. Culliney, "The transduction method—Design of logic networks based on permissible functions," *IEEE Trans. Comput.*, vol. 38, pp. 1404–1424, 1989.
[13] A. Saldanha, A. R. Wang, R. K. Brayton, and A. Sangiovanni-Vincentelli, "Multi-level logic simplification using don't cares and filters," in *Proc. Design Automation Conf.*, 1989.
[14] H. Savoj and R. K. Brayton, "The use of observability and external don't cares for the simplification of multi-level networks," in *Proc. Design Automation Conf.*, 1990.
[15] A. Stoelzle, S. Narayanaswamy, K. Kornegay, H. Murveit, and J. Rabaey, "A VLSI wordprocessing subsystem for a real time large vocabulary continuous speech recognition system," in *Proc. Custom Integrated Circuits Conf.*, 1989.
[16] H. Touati and R. K. Brayton, "Computing initial states of retimed circuits," unpublished.
[17] T. Villa, "NOVA: State assignment for finite state machines for optimal two-level logic implementations," in *Proc. Design Automation Conf.*, 1989.

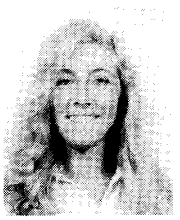
*



Sharad Malik received the B. Tech. degree in electrical engineering from the Indian Institute of Technology, New Delhi, India in 1985 and the M.S. degree in computer science from the University of California, Berkeley in 1987. Currently he is a doctoral student in Computer Science at U.C. Berkeley specializing in design automation for VLSI systems.

His current interests are in the synthesis and verification of digital systems.

Mr. Malik received the President of India's Gold Medal in 1985.



Ellen M. Sentovich received the B.S. and M.S. degrees in electrical engineering from the University of California, Berkeley in 1985 and 1988, respectively. She is currently working towards the Ph.D. degree in electrical engineering at the University of California, Berkeley.

Her current research interests include multilevel logic synthesis and sequential logic synthesis, optimization, and verification.

*

Robert K. Brayton (M'75–SM'78–F'81) received the B.S.E.E. degree from Iowa State University in 1956 and the Ph.D. degree in mathematics from MIT in 1961.

From 1961 to 1987 he was a member of the Mathematical Sciences Department of the IBM T. J. Watson Research Center at Yorktown Heights, N.Y. In 1987 he joined the Department of Electrical Engineering and Computer Sciences at Berkeley, where he is a Professor. He has authored or coauthored more than 100 technical papers, and is the co-author of two books, *Logic Minimization Algorithms for VSLI Synthesis* and *Computer Aided Design: Sensitivity and Optimization*. During the years 1965–1966 he was a visiting Professor at MIT, 1975–1976 at Imperial College, London, and 1985–1986 at UC Berkeley. From 1970 to 1972, he was the assistant director of the IBM Mathematical Sciences Department, and in recent years a second-level manager for mathematical algorithms in Logic Design, Mathematical Programming, General Mathematical Analysis, and Parallel Computing. During 1970-1972, he was a NSF Chatauqua Lecturer on Mathematical Models and Computing. He has been a member of the National Science Foundation Advisory Panel in Mathematics and has served on various IEEE committees, including the Circuits and Systems ADCOM, CANDE, and Large Scale Systems. He is currently an associate editor for the IEEE TRANSACTIONS ON COMPUTER-AIDED DESIGN. His interests and contributions have been in the areas of nonlinear networks, stability theory, numerical methods for differential equations, sparse matrices, simulation of electrical circuits, optimization methods for circuit design, logic synthesis and minimization, silicon compilers, and general CAD/VLSI issues.

Dr. Brayton is a fellow of the AAAS. He received best paper awards from the IEEE Circuits and Systems Society in 1971 and 1987, and has received four IBM Outstanding Innovation Awards and two IBM patent awards.

*

Alberto Sangiovanni-Vincentelli (M'74–SM'81–F'83) for a photograph and biography please see page 3 of this issue.