Performance Optimization of Pipelined Logic Circuits Using Peripheral Retiming and Resynthesis

Sharad Malik, Kanwar Jit Singh, Robert K. Brayton, *Fellow, IEEE*, and Alberto Sangiovanni-Vincentelli, *Fellow, IEEE*

Abstract—**We consider the problem of minimizing the cycle time of a given pipelined circuit. Existing approaches are suboptimal since they do not consider the possibility of simultaneously resynthesizing the combinational logic and moving the latches using retiming. In [10] the idea of simultaneous retiming and resynthesis was introduced. We use the concepts presented there to optimize a pipelined circuit to meet a given cycle time. An instance of the pipelined cycle optimization problem is specified by the circuit, a set of input arrival times relative to the clock, a set of output required times relative to the clock, and a given cycle time that it must meet. Given the instance of the pipelined performance optimization problem we construct an instance of a combinational speedup problem. This is specified by a combinational logic circuit, a set of arrival times on the inputs, and a set of required times for the outputs which must be met. We then give a constructive proof that the pipelined problem has a solution if and only if the combinational problem has a solution. This result is significant since it shows it is enough to consider only the combinational speedup problem and all known techniques for that (e.g., [12], [13]) can be directly applied to generate a solution for the pipelined problem.**

I. Introduction

In order to be completely accepted by IC designers, behavioral and logic synthesis tools must satisfactorily address the issue of meeting performance constraints. This area has been largely ignored in the past; the emphasis thus far has been on area optimization. Design failures that result from ignoring performance requirements during synthesis can be counterproductive to the very use of synthesis since timing problems take a long time to fix. We feel that the modeling and consideration of performance issues must be taken into account explicitly during each stage of the design synthesis.

Previous work in this area can be classified into two categories. In the first, only resynthesis of individual blocks of combinational logic is considered. These include critical-path resynthesis (e.g., [3], [12]), gate decomposition (e.g., [11]), technology mapping (e.g., [5, 13]), and buffer optimization (e.g., [2]). Since these do not consider modifying the position of latches or flip-flops, an initial bad placement of the latches restricts the possible improvements. The second category includes techniques that reposition the latches so as to get a shorter cycle time. The technique of retiming, developed by Leiserson *et al.* [6]–[8], provides a polynomial time procedure to determine the optimum position of latches in a single-phase, edge-triggered sequential circuit. However, no effort is made here to modify any of the combinational logic.

The crucial element in combining the techniques of retiming and resynthesis is the decision as to when, where, and in what order to apply these operations. Some preliminary work in combining them has been done in [1], but the approach there is heuristic driven with no optimality guarantees. In [4] retiming was combined with a restricted class of combinational optimization techniques with limited success. In this paper we provide a rigorous solution to this problem for a special yet important class of circuits, *viz* pipelined circuits.[1] Here we transform the problem of speeding up a pipeline circuit into one of resynthesizing a combinational logic circuit with appropriate timing constraints. This is achieved by peripheral retiming [10] which moves the latches to the inputs and outputs of a circuit. The resulting maximal combinational sub-circuit can be subject to any delay-reducing transformation in an effort to meet the timing constraints specified for it. The timing constraints are based on the cycle time that the user desires. We show that if the timing constraints on the maximal combinational sub-network are met, then it is always possible to retime the resynthesized circuit to meet the desired cycle time. We also show that any circuit that meets the cycle-time constraint can be obtained by peripheral retiming, appropriate resynthesis, and then retiming. These two facts are used to show that the proposed method is the best way to resynthesize a pipeline circuit to meet its performance (cycle time) target.

This paper is organized as follows. Section II reviews the results in retiming and peripheral retiming upon which the work presented in this paper is based. Section III introduces the topology of pipelined circuits as the class of circuits that can be peripherally retimed. Next, in Section

Manuscript received August 20, 1991; revised April 29, 1992. This paper was recommended by Associate Editor K. Keutzer.

S. Malik was with the Department of Electrical Engineering and Computer Science, University of California, Berkeley, CA. He is now with the Department of Electrical Engineering, Princeton University, Princeton, NJ 08544-5263.

K. J. Singh, R. K. Brayton, and A. Sangiovanni-Vincentelli are with the Department of Electrical Engineering and Computer Science, University of California, Berkeley, CA 94720.

IEEE Log Number 9205882.

[1] We consider a single-phase, edge-triggered design methodology that is used in a large number of ASIC applications and leads itself to polynomial time analysis and retiming algorithms.

IV, the application of these ideas to the performance optimization of pipelined circuits is presented. Section V presents experimental results of applying these ideas to example circuits. Finally, in Section VI, a summary of the contributions of this paper and directions for future work are presented.

II. Retiming and Peripheral Retiming: A Review

This section reviews the basic ideas in retiming and peripheral retiming. This material is based on the work presented in [7], [10], [9]. The proofs for all the theorems in this section have been omitted; they can be found in [10].

2.1. Abstraction of Acyclic Circuits

The discussion in this section is focused on circuits whose underlying topology is acyclic. (These are also referred to as *feed-forward circuits*.) These circuits are modeled by a directed acyclic graph called a *communication graph*[2] where each vertex v represents either

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

The input/output pins correspond to the primary inputs and primary outputs of the circuit. The granularity of the combinational logic block may vary: it may be a single gate or a larger module such as an adder. The vertices in the graph are connected by directed edges. A restriction is placed so that each input pin has no incoming edges and exactly one outgoing edge (a single-output source), and that an output pin has no outgoing edges and exactly one incoming edge (a single-input sink). If a primary input is used in more than one place in the circuit, then this is captured by introducing a dummy vertex in the graph that handles the multiple out-edges. (The out-edges are also referred to as fan-out.) 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 latches between the two vertices it connects. An example of a sequential circuit and its communication graph is shown in Fig. 1. Note that the multiple fan-out of primary input a is handled through the internal vertex a' in the graph. For simplicity in the figure, if the edge weight is 0, then the edge label is omitted. A sequential circuit is alternatively referred to as a sequential network. The terms circuit, network, and graph are used interchangeably whenever there is no ambiguity.

A *path* between two vertices v_1 and v_2 in the graph is a sequence of consecutive 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 b to output f has weight 0, while the path from b to e has weight 1.

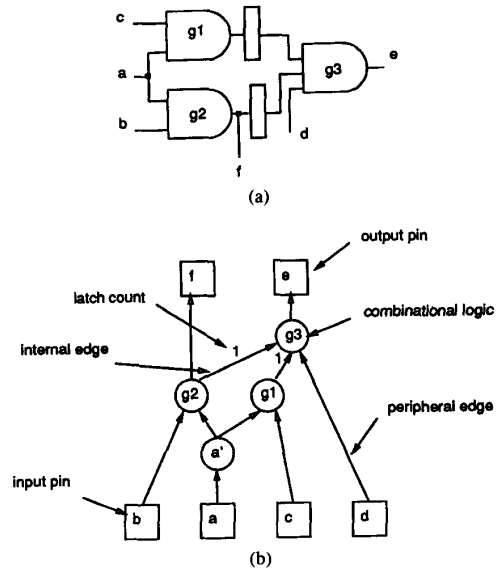


Fig. 1. Examples: (a) Sequential circuits. (b) Communication graph.

2.2. Retiming

The cycle time of a synchronous sequential circuit is determined by the length of the longest path between any two latches in the circuit. The concept of *retiming* exploits the ability to move the latches in the circuit in order to decrease the length of the longest path in the circuit while preserving its functional behavior. Retiming algorithms were first proposed by Leiserson *et al.* [7], [6]. To illustrate this with a small example, consider Fig. 2. The circuit on the left is functionally equivalent to the circuit on the right since delaying the output of gate g by a cycle is equivalent to delaying each of its inputs by a cycle. The movement of latches during retiming is quantified by an integer $L(v)$ (called the lag of v) for each vertex v, which represents the number of latches that are to be moved in the circuit from each out-edge of vertex v to each of its in-edges. Thus, in Fig. 2 the circuit on the right is obtained from the circuit on the left by retiming g by $+1$. Similarly, in obtaining the circuit on the left from that on the right g has been retimed by -1. For input and output pins the lag is 0. Consider an edge $e(u, v)$ in the circuit. Let $w(e)$ be the weight of the edge in the graph before retiming and $w_r(e)$ be the weight after retiming. w_r is determined from w and the lags by the following equation:

$$w_r(e) = w(e) + L(v) - L(u).$$

The following definition is from [7].

Definition 2.1: A legal retiming *is the assignment of an integer $L(v)$ to each vertex in the communication graph such that for each edge e, $w_r(e) \geq 0$.*

For a legal retiming, the edge weights of the retimed graph must be non-negative; indicating a non-negative

[2]This is related to the definition of a communication graph presented in [7].

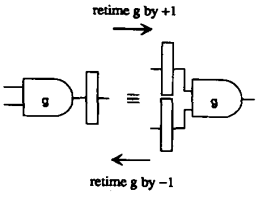


Fig. 2. Retiming.

number of latches on each edge. Thus, there exists a real physical circuit corresponding to this graph. This is not possible with negative edge weights in the retimed circuit since there is no physical circuit component corresponding to a negative latch. A legal retiming has been shown [7] to generate a circuit that is functionally equivalent to the original circuit. Fig. 3 shows a legal retiming on the communciation graph of Fig. 1. Here, the lag of $g1$ is $+1$ and the lag for all other vertices is 0.

2.3. *Extensions to Retiming*

The use of retiming can be extended by introducing the concept of a "negative" latch, i.e., an edge weight in the graph that is negative. Negative edge weights are permitted on peripheral edges only. Allowing a negative edge weight n on a peripheral edge is equivalent to "borrowing" n latches from the environment. The latches may be "returned" by a subsequent retiming step, whereby n latches 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. This is illustrated with the circuit in Fig. 4(a). Consider the latch on the connection between $g2$ and $g3$. In order to move this latch from its present position either $g3$ is retimed by -1 (for forward motion) or $g2$ retimed by $+1$ (backward motion). If $g3$ is retimed by -1, this will result in an edge weight of -1 at input d. If $g2$ is retimed by $+1$, this will result in an edge weight of -1 at output f. Thus, neither of these retimings is legal. However, if this "illegal" retiming is permitted temporarily, then it is possible to gain additional advantage over what is permitted by just legal retimings. Fig. 4(b) shows the circuit after $g3$ has been retimed by -1. The edge weight of -1 on input d is represented by the latch in dotted lines and with the label -1 on it. At this point, all the gates in the circuit are part of a single combinational logic block. For this logic block, it can be shown that the functionality remains unchanged if the connection from a to $g1$ is deleted; i.e., this connection is redundant or in testing parlance, untestable for a stuck-at-1 fault in the context of this combinational logic block. Thus, this connection may be deleted and $g1$ replaced by a wire. This simplified circuit is shown in Fig. 4(c). Note that this connection (from a to $g1$) is not redundant in the context of the original combinational block (consisting of $g1$ and $g2$) defined by the original position of latches. Only after we could view all the gates as part of a single combinational logic block

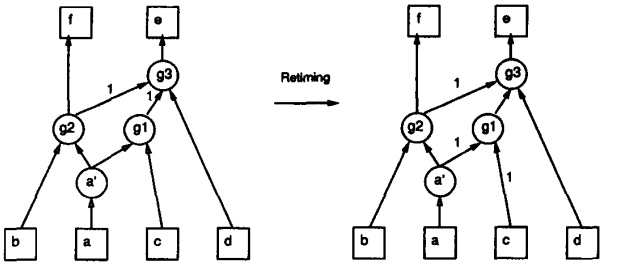


Fig. 3. Legal retiming.

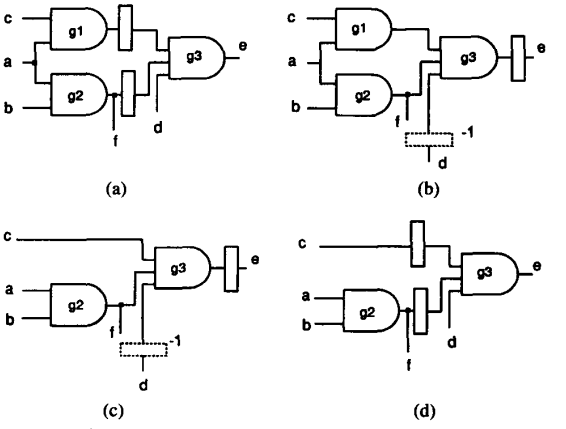


Fig. 4. Use of a negative latch. (a) Original circuit. (b) Circuit after $g3$ is retimed by -1. (c) Simplified circuit after redundant connection is deleted. (d) Circuit after $g3$ is retimed by $+1$, annihilating the negative latch at input d.

was this redundancy exposed. Of course, this circuit is still not realizable since there is a negative latch at input d. This situation can easily be rectified by retiming $g3$ by $+1$. This annihilates the negative latch at input d resulting in the circuit in Fig. 4(d). This example illustrates the advantage gained by permitting illegal retimings temporarily. Later in this section it is shown why this is a legitimate operation.

Let us now go back and see what enabled us to eliminate $g1$ in the previous example. Once we were able to consider all three gates in the circuit as part of a larger combinational block, we could use the combinational optimization technique of redundancy removal to detect and delete the redundant connection. This is precisely what we were looking for, i.e., a way to consider and exploit logical relationships between gates that extend beyond latch boundaries. Ideally, we would like to push out all the latches in the circuit to the peripheral edges. This results in no latches on any of the internal edges and thus all the gates are part of the same combinational logic block. This permits the use of any combinational logic optimization technique on this larger combinational block. The notation of a *peripheral retiming* does precisely this.

Definition 2.2: A peripheral retiming *is a retiming such that for each internal edge* e, $w_r(e) = 0$.

This is graphically shown in Fig. 5. After peripheral

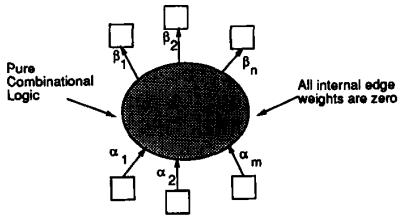


Fig. 5. Peripheral retiming.

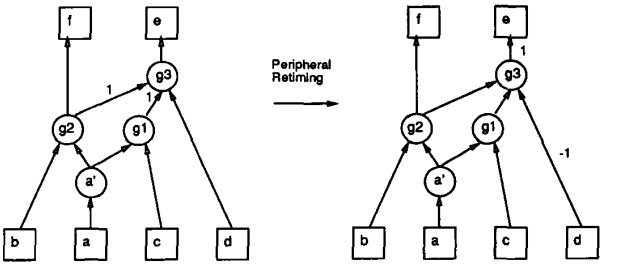


Fig. 6. Peripheral retiming.

retiming, there are α_i latches at input pin i, β_j latches at output pin j, and no latches on any internal edge.

The circuit in Fig. 4(b) is a peripheral retiming of the one in Fig. 4(a). The same peripheral retiming is shown in terms of the communication graph in Fig. 6.

The condition that $w_r(e)$ is 0 for all internal edges forces all latches to the peripheral edges. Note that the definition permits negative weights on the peripheral edges, which corresponds to the negative latch concept presented earlier. Permitting negative latches temporarily on peripheral edges is a legitimate operation as shown by the following theorem. Functional equivalence here refers to the equivalence of the finite automata corresponding to the initial and final circuits.

Theorem 2.1: A circuit that undergoes a peripheral retiming, combinational optimization, and a subsequent legal retiming is functionally equivalent to the original circuit.

2.4. Conditions for Peripheral Retiming

Not all circuit topologies permit a peripheral retiming. Two such topologies are now considered.

Consider the circuit in Fig. 7 and its corresponding communication graph. All attempts to move the latch (either backward or forward) to the periphery result in a negative weight on the edge between b' and $g2$. In fact, as will be shown later in this section, this circuit cannot be peripherally retimed. Examining the circuit gives us some insight into why this is so. The output c depends on the value of input b at two different times. Let us assume that a peripheral retiming were possible. Then in the peripherally retimed circuit c would depend only on one time value of b since all paths from b to c would have the same number of latches, *viz* $\alpha_a + \beta_c$. This would not capture the correct behavior and is the reason why no peripheral retiming exists.

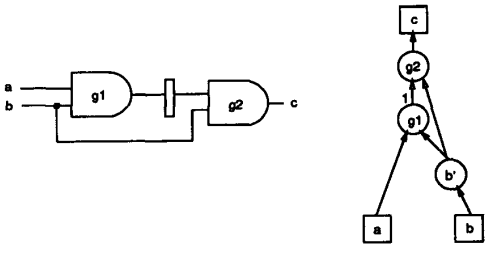


Fig. 7. Circuit with no peripheral retiming: Example 1.

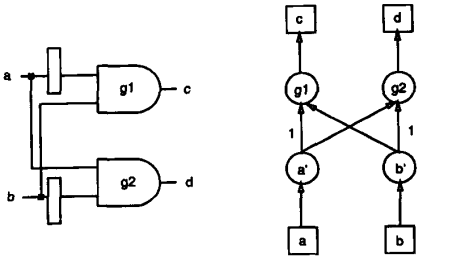


Fig. 8. Circuit with no peripheral retiming: Example 2.

Now consider the circuit and communication graph in Fig. 8. Again, it is not possible to move the latches to the periphery without introducing a negative weight on the internal edges $(a', g2)$ and $(b', g1)$. The intuition as to why a peripheral retiming does not exist here is more complicated than in the previous case. For output c, inputs a and b are delayed by 1 and 0 cycles, respectively. For output d, these inputs are delayed by 0 and 1 cycles, respectively. Let us assume a peripheral retiming were possible. Then c would see a and b delayed by $\alpha_a + \beta_c$ and $\alpha_b + \beta_c$, respectively. Similarly, d would see them delayed by $\alpha_a + \beta_d$ and $\alpha_b + \beta_d$, respectively. The only difference in delays for the inputs that d sees with respect to c is due to the different number of latches on the peripheral edges at c and d, i.e., $\beta_d - \beta_c$. Thus, the input delays are the same for all the outputs except for a constant offset that depends on the output, and this offset is added to each of the input delays. Clearly this is not possible for this example.[3]

While these two examples give some insight into when a peripheral retiming may not exist, by themselves they do not provide a characterization of circuits that permit a peripheral retiming. In order to obtain such a characterization, the *path weight matrix* of a network is defined.

Definition 2.3: A path weight matrix, W, of a sequential network is an $m \times n$ matrix, where

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,

[3]Even though the circuits in Figs. 7 and 8 cannot be peripherally retimed, sub-circuits of these circuits can.

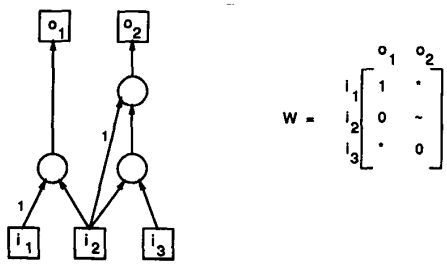


Fig. 9. Path weight matrix.

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.

Fig. 9 shows a communication graph and the corresponding path weight matrix.

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

Definition 2.4: A matrix W is *satisfiable* if

a) $W_{ij} \neq \sim, \forall i, j$,
b) $\exists \alpha_i, \exists \beta_j$, $1 \leq i \leq m$, $1 \leq j \leq n$, $\alpha_i, \beta_j \in I$ such that for each $W_{ij} \neq *$, $W_{ij} = \alpha_i + \beta_j$.

These two conditions have been motivated by the two examples considered above. Cyclic circuits (assuming there is at least one latch on each path containing a cycle) will have $W_{ij} = \sim$ for each path containing a cycle. Thus, only acyclic communication graphs can have a satisfiable path weight matrix.

The path weight matrix helps us derive the necessary and sufficient conditions on a circuit for it to have a peripheral retiming.

Theorem 2.2: A sequential network has a peripheral retiming if and only if its path weight matrix is satisfiable.

Note the significance of this result: it gives a complete characterization of the class of sequential circuits for which all the latches can be pushed to the periphery; i.e., it specifies the necessary as well as the sufficient conditions on the circuit topology.

A peripheral retiming involves finding a set of α's and β's that satisfy the path weight matrix. These α's and β's specify the peripheral retiming. In the retimed circuit there are α_i latches at the ith input pin and β_j latches at the jth output pin. A matrix that is satisfiable has no $\sim$ entries, and has at least one set of α_i's and β_j's such that $\alpha_i + \beta_j = W_{ij}$.

III. Peripherally Retimable Circuits: General Topology

Theorem 2.2 states the necessary and sufficient conditions under which a circuit has a peripheral retiming. However, it gives no feel for the general topology of these circuits. In order to understand them better we would like to characterize them in terms of their general topology.

Fig. 10(a) shows the general topology of circuits that permit a peripheral retiming. A peripherally retimed circuit is shown in Fig. 10(b). This has been obtained by moving the latches forward through the circuit, borrowing latches at the inputs when required. Circuits satisfying this topology are called *balanced* or *pipelined* circuits. Note that inputs and outputs are permitted to and from each stage of the pipeline. This is more general than simple pipelines, where data enters the first stage and the result leaves the last stage. Of course any of the input or output vectors I_i or O_j may be empty.

It is now shown that this is exactly the topology corresponding to circuits that can be peripherally retimed. Let C be a circuit that can be peripherally retimed and C_r be the peripherally retimed circuit. We need to show the following.

1) In C_r, α_i is non-positive for all i and β_j is non-negative for each j.
2) There is no path from input i to output j such that $\alpha_i + \beta_j < 0$.

Let α_i latches be an input i and β_j latches be at output j in C_r. This can be modified to obtain another solution as follows: Let $\alpha_{\max}$ be the value of the largest α_i. Subtract $\alpha_{\max}$ from each α_i and add it to each β_j. The resulting values of α's and β's also satisfy the path weight matrix. As in Fig. 10(b), C_r has non-positive α's and non-negative β's. To see that the β's must be non-negative, assume that this were not true, i.e., some β_j were negative. Let i be an input from which there is a path to output j. Then the path weight in C_r from i to j must be negative since α_i is non-positive. This cannot happen since in the initial graph, and hence the retimed graph, there is no path with negative weight. Thus, all β's are nonnegative. Another consequence of the fact that there is no path in the initial graph of negative weight is that there cannot be a path from input i to output j such that $\alpha_i + \beta_j < 0$ in the retimed circuit. Thus, C_r satisfies both the conditions listed above and its topology is precisely that of Fig. 10(b). Therefore, C must have the topology of Fig. 10(a).

The circuit in Fig. 10(a) naturally suggests that the latches are pipelined latches and that there is an underlying combinational logic block. Peripheral retiming exposes this by moving the latches to the periphery. It may seem that the combinational logic can be exposed by just ignoring the latches (replacing them by wires). This certainly is true. However, once this circuit has been resynthesized, the latches need to be placed back in the circuit. This needs to be done while guaranteeing that each input and output is in the correct stage of the pipeline. Peripheral retiming handles this elegantly. The latches at the periphery are place holders for this information. When the circuit is retimed after resynthesis, retiming ensures that each input and output lies in the correct stage in the pipeline. The significance of negative latches is reiterated;

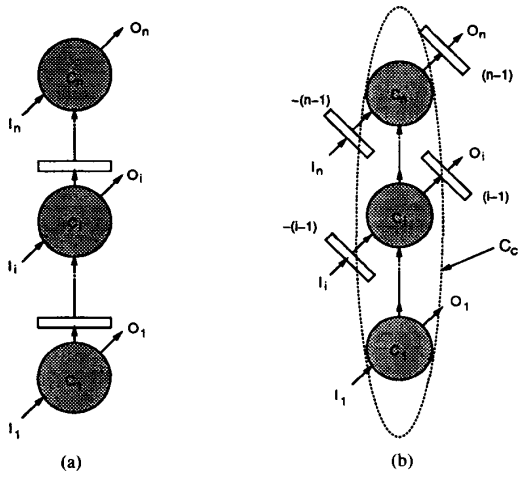


Fig. 10. Pipelined circuits and their retiming. (a) Pipelined circuit. (b) Peripherally retimed circuit.

without them it would not be possible to handle inputs and outputs from each stage.

IV. Performance Optimization

The three-step procedure of peripheral retiming, combinational resynthesis, and a final legal retiming to exploit combinational optimization techniques beyond latch boundaries was described in Section II. In the example presented there, combinational optimization was being used for area reduction. However, the retiming and resynthesis framework does not specify the nature of the resynthesis permitted; any combinational resynthesis is valid. This section examines the use of performance-directed resynthesis in this framework.

4.1. Two Problems in Performance Optimization

4.1.1. Pipelined Performance Optimization

Pipelined circuits were introduced in Section III. Recall that a pipelined circuit of n stages consists of n combinational circuits ($C_1, \cdots, C_n$) with stage i communicating with stage $i + 1$ through some signals that are latched. (See Fig. 10(a).) Each C_i may have inputs I_i and outputs O_i besides the latched inputs and outputs used to communicate with adjacent pipeline stages. This description of a pipeline is general since it does not restrict all inputs to come in at the first stage and outputs to leave the last stage. Allowing inputs to any stage (rather than just the first stage) in the pipeline provides the following additional flexibility:

1) The pipeline can operate on multiple streams of data which may arrive separated from each other by an arbitrary number of clock cycles.

2) The pipeline can consider control signals that arrive in cycles subsequent to the initial data and perform the remaining computation based on these.

Similarly, allowing outputs from any stage permits the following:

1) Multiple computations may be performed with the different results being available during different clock periods.

2) Error conditions or any status signals can be provided as outputs in the cycle they are computed.

This description of pipelined circuits is general and includes most circuits that are considered to be pipelined by digital circuit designers.

The performance optimization problem of pipelined circuits is to maximize the clocking rate or equivalently minimize the cycle time of the circuit. This determines the throughput of the circuit. Sometimes the cycle time, c, that needs to be attained is specified as a constraint, based on the requirements of the system of which this circuit is a part. In that case, the optimization problem is to meet the specified cycle time constraint. In general, not all inputs are available at the clock edge; for example, they may arrive later because of communication delays. Similarly some of the outputs may be required well before the clock edge, say, in order to satisfy setup time requirements. Thus, with each input signal an arrival time, a, is associated which is the time after the clock edge that the signal is available, and with each output signal a required time, r, is associated which is the time before the clock edge that this signal must be ready. Let A and R be the sets of arrival times and required times for the inputs and outputs, respectively. These capture the timing constraints due to the environment of this circuit. Thus, an instance of the pipelined performance optimization problem $\mathcal{P}_P$, is specified as $\mathcal{P}_P = \{\mathcal{C}, c, A, R\}$, where $\mathcal{C}$ is the circuit and c the desired cycle time constraint. Note that this problem statement assumes that c is provided as part of the problem. If the problem is to minimize c, this can be done by solving a number of problem instances $\mathcal{P}_P$, each with a known c, by performing a binary search for the least feasible c.

4.1.2. Combinational Speedup

The combinational speedup problem has been well studied in recent years. Here, a given combinational circuit, $\mathcal{C}$, is to be resynthesized so that it meets its timing constraints. The timing constraints are specified as required times, r, on the outputs of the combinational circuit. In this case these times are absolute and not with reference to any clock edge. As before, the inputs may be arbitrarily delayed and an absolute arrival time a is associated with each input. Thus, an instance of this problem, $\mathcal{P}_C$, is specified as $\mathcal{P}_C = \{\mathcal{C}, A, R\}$.

4.1.3. Problem Transformation

Given an instance of the pipelined performance optimization problem, $\mathcal{P}_P$, it is first transformed to an instance of the combinational speedup problem $\mathcal{P}_C$. Subsequently

it is demonstrated how a solution of $\mathcal{P}_C$ may be retimed to obtain a solution of $\mathcal{P}_P$.

As described in Section III we obtain a peripheral retiming for the pipelined circuit by retiming block C_i by $-(i-1)$. (See Fig. 10(b).) This is not unique; several possible peripheral retimings exist. In terms of the circuit this implies that we sweep all the latches forward through the circuit, borrowing latches at the inputs as needed. This results in the peripherally retimed circuits shown in Fig. 10(b). As before, peripheral retiming has exposed a larger combinational circuit, $\mathcal{C}_C$, which is the cascade of $C_1, C_2, \cdots, C_n$. This is the combinational circuit that we will use for the transformed problem, $\mathcal{P}_C$. In order to complete the definition of $\mathcal{P}_C$ we need to specify A and R. We determine them by using c, A, and R from $\mathcal{P}_P$ as follows. For an input to stage i of $\mathcal{C}$ with arrival time a in $\mathcal{P}_P$, the arrival time in $\mathcal{P}_C$ is $a + (i-1)c$. For an output of stage i in $\mathcal{C}$ with required time r in $\mathcal{P}_P$, the required time in $\mathcal{P}_C$ is $ic - r$. Note that it is easy to determine which stage an input or an output belongs to by looking at the number of latches in the peripherally retimed circuit at that input or output. The intuition behind this choice of modifications of arrival and required times is that it captures the fact that the input arrives only after $i-1$ cycles and the output is required only before the ith cycle. In the next section we will see how this choice of A and R results in an interesting relationship between problems $\mathcal{P}_P$ and $\mathcal{P}_C$.

4.2. Main Results

We would like to obtain a solution of $\mathcal{P}_P$ from a solution of $\mathcal{P}_C$. It is not immediately obvious that this is possible; in fact, it seems almost unlikely to happen as the analysis below shows. However, we need a few simple definitions first.

Definition 4.1: A *path* in a circuit is an alternating sequence of consecutive connections (possibly latched) and gates.

As in [6], [7] a propagation delay, d_g, is associated with each gate in the circuit.

Definition 4.2: A *segment* of a path is a path from an input or a latch to an output or a latch.

Definition 4.3: The delay of a segment, s, is the sum of the gate delays along the segment and is denoted by D_s.

Definition 4.4: The lumped delay of a segment s, denoted by λ_s, is the sum of the combinational logic delays associated with the gates and the times associated with late arrival of an input or early requirement of an output. Thus, $\lambda_s = a + D_s + r$.

Of course, a or r are equal to 0 if the segment has no input or output, respectively.

Definition 4.5: A latch is said to be *forward-blocked* if it cannot be legally moved forward across its output gate, g, without violating a cycle time constraint.

This may happen for two reasons.

1) There exists a path from an input to g which does not have any latch on it. Thus, it is never possible to have a latch at each input of g which is needed for the legal forward motion of the latch. (See Fig. 11(a).)
2) There exists a path from a critical latch to g. Informally a latch is critical if it cannot be moved forward without violating the cycle time constraint. (The formal definition of a critical latch is deferred till later in this section.) Thus, it is not possible to move a latch from each input of g to its output. (See Fig. 11(b).)

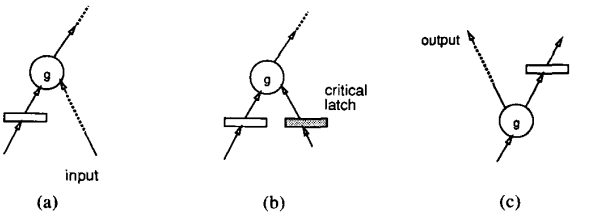


Fig. 11. Blocked latch motion. (a) Forward-blocked latch: There exists a path from an input to g with no latch. (b) Forward-blocked latch: There exists a path from a critical latch to g. (c) Backward-blocked latch: There exists a path from g to an output without a latch.

Definition 4.6: A latch is said to be *backward-blocked* if it cannot be legally moved backward through a gate g.

For this to happen, there must be a path from g to an output which does not have a latch on it. Thus, it is never possible to have a latch at each output of g, which is needed to move this latch backward. (See Fig. 11(c).)

Consider a path, p, from an input of stage i to an output of stage $i+j$ in $\mathcal{P}_C$. Assume that the solution of $\mathcal{P}_C$ just meets the delay constraints for this path; i.e., the path is critical for $\mathcal{P}_C$. Therefore, the combinational logic delay, $delay(p)$, along this critical path is given by

$$delay(p) = ((i+j)c - r) - (a + (i+1)c) - \epsilon$$

where ϵ is an arbitrary small positive number. Simplifying this results in

$$delay(p) = (j+1)c - r - a - \epsilon. \tag{1}$$

Now suppose that we do find a final retiming that meets the cycle time constraint c along this path. Since retiming does not change the number of latches between any input and output, there will be j latches between the input-output pair in consideration. Thus, in the retimed circuit this path has $j+1$ segments. Then for each segment, $\lambda_k < c$. Summing over k for this path, we see that

$$\sum_{k=0}^{j} \lambda_k < (j+1)c$$

which gives

$$a + \sum_{k=0}^{j} D_k + r < (j+1)c$$

or equivalently,

$$\sum_{k=0}^{j} D_k < (j+1)c - r - a.$$

But $\Sigma_{k=0}^{j} D_k$ is *delay*(p) and we know from (1) that this is equal to $(j+1)c - r - a - \epsilon$. Thus, we cannot add any delay to any segment without violating the cycle constraint. Therefore, each path segment along this path is critical inasmuch as moving any latch would violate the cycle constraint. It is possible that the solution to $\mathcal{P}_C$ may actually result in the constraints along all paths in $\mathcal{C}_c$ to be just met. In that case, for each path there is only one available position of latches that will meet the cycle time constraint. Since different paths overlap, it is possible that the positions of latches dictated by each path may be in conflict.

To make matters worse, latches cannot be arbitrarily placed along a path as assumed by the above analysis. The latch motion may be blocked in either the forward or backward direction. This makes it even less probable that the latches can be positioned so that for all the paths they lie at the single position that meets the cycle time constraint and simultaneously avoid being blocked by the positions of inputs and outputs.

Since there are only discrete positions along the length of the path where a latch may be placed, *viz* before and after gates, the granularity of control that we have over the latch positions is only the largest possible gate delay δ. To handle this, the following relaxed problem is defined: $\mathcal{P}'_P = \{\mathcal{C}, c+\delta, A, R\}$ derived from $\mathcal{P}_P$. A solution to $\mathcal{P}'_P$ exceeds the cycle time constraint for $\mathcal{P}_P$ by no more than δ.

To show that any solution of $\mathcal{P}_c$ can be retimed to get a solution of $\mathcal{P}'_P$ a few more concepts must be considered.

Definition 4.7: A segment is said to be *critical* if it just meets the performance constraints for the segment; i.e. the latch at the end of the segment cannot be moved forward across a gate g without violating the constraints for $\mathcal{P}'_P$. Thus, $c \le \lambda_k < c+\delta$ and $\lambda_k + d_g \ge c+\delta$ for a critical segment.

Definition 4.8: A segment is said to be *violating* if it does not meet the timing requirements for $\mathcal{P}'_k$. Thus, $\lambda_k \ge c+\delta$ for a violating segment.

Definition 4.9: A path is said to be critical if all its segments are critical.

Definition 4.10: A latch is said to be critical if it terminates a critical segment.

Definition 4.11: A path is said to be violating if the last segment is violating and all the other segments are critical.

We label each input of the circuit based on the number of latches at that input in the peripherally retimed circuit in Fig. 10(b). If an input has $-(i)$ latches, then its label is i. Each latch in a pipelined circuit can be labeled by a unique integer that specifies its level in the circuit. This labeling obeys the following two rules.

1) If there is a purely combinational path (with no latches) from an input with label i to this latch, then the label on the latch must be i.
2) If there is a path with no latches from a latch with label $i-1$ to this latch, then its label must be i.

For a pipelined circuit, a unique labeling exists for the latches which satisfies both rules.

The following basic lemma is critical to prove the equivalence of $\mathcal{P}'_P$ and $\mathcal{P}_C$.

Lemma 4.1: Let k be any integer not exceeding the largest label on any latch in $\mathcal{C}$. Then, it is possible legally to retime $\mathcal{C}$ to $\mathcal{C}_r$, such that in $\mathcal{C}_r$:

1) There exists a violating path from an input to an output that passes through latches labeled only $< k$, OR
2) The following three conditions are satisfied:
   (a) No segment starting at a latch or input labeled $< k$ is violating.
   (b) Each latch labeled $\le k$ is either critical or forward-blocked.
   (c) Each critical latch terminates a critical path from some input to that latch.

Proof: The proof is by induction on k.

Induction Hypothesis: Assume the statement in the lemma is true for all $i < k$.

Induction Basis: The statement in the lemma is proven for $i = 0$.

Let us try and place all latches (using retiming) with label 0 so that the input segments do not violate the cycle time constraint $c+\delta$.

Case 1: It is not possible to do so. Then one of the following must be true.

1) There exists a path from an input to an output for which $a + D + r \ge c + \delta$. This path is a violating path from an input to an output and thus the first condition in the lemma is satisfied.
2) There exists a path from an input, i, to a latch, l, for which $a + D \ge c + \delta$. This latch must be backward-blocked or else we would have moved it backwards to get rid of this violation. If it is backward blocked, then the input gate of this latch must have a path to an output, o, with no latch on it. Thus, for the path from i to o, the following must be true: $a + D + r \ge c + \delta$. This follows from the fact that r is non-negative and you need to go through at least the same, and possibly additional logic while going from i to o instead of going from i to l. This input to output path is violating and the first condition in the lemma is satisfied.

Case 2: It is possible to do so. Then we can move each latch with label 0 forward till it is either critical or forward-blocked. If a latch is critical, then it must be so because it terminates a critical path from some input. Now all three parts in the second condition of the lemma statement are satisfied.

Induction Step: It is now proven for $i = k$. As in the basis step we can either place all latches with label k without violating the cycle time constraint $c+\delta$, or we cannot. Consider each of these separately.

Case 1: We cannot. Then by an argument similar to that used in Case 1 of the base case, there is a segment

from either a latch or an input or an output that is violating. In fact, this violating segment must be from an input or a critical latch. To see why this must be so, suppose that the violating segment is from a non-critical latch to the output. Since the non-critical latch was blocked by either a critical latch or an input, the segment from this critical latch or input to the output is violating. If it is from an input, then the path from this input to the output is violating, satisfying the first condition of the lemma. If it is from a critical latch, by the induction hypothesis the critical latch terminates a critical path from an input. This path along with the violating segment starting from this latch forms a violating path from an input to an output, satisfying the first condition in the lemma.

Case 2: We can. Then each latch with label k can be moved forward until it is either forward-blocked or critical while ensuring that no segment is violating. For each critical latch with label k there must be a critical segment from an input or a critical latch. The reasoning as to why all critical segments to a latch cannot be from non-critical latches is the same as that used in Case 1 above. If the critical segment is from an input, then the critical latch does terminate a critical path from an input. If it is from a critical latch, then by the induction hypothesis this critical latch terminates a critical path starting at an input. The concatenation of this critical path with this critical segment gives the required critical path from an input for the critical latch with label k which satisfies the second condition in the lemma. □

With this we can now prove the following.

Lemma 4.2: If the solution to $\mathcal{P}_c$ cannot be retimed to meet cycle time constraint $c + \delta$, then there must exist a violating path from an input to an output.

Proof: Let k be the maximum label on a latch in any retimed circuit. Using Lemma 4.1 with this k, we see that either there is a violating path from an input to an output, in which case we are done, or all critical latches with label k terminate a critical path starting at an input. Note that since the cycle time constraint was not met, there must be a violating segment starting at a latch or input with label k. Actually this violating segment must be from a critical latch or input with label k by the same argument used in Case 2 of the induction step in the proof for Lemma 4.1. If this is from an input, then this segment forms a violating path from an input to an output. If it is from a critical latch, then this segment appended to the critical path from an input terminating at that latch forms a violating path from an input to an output. □

Finally, it is shown that the existence of a violating path implies that the constraint for that path in $\mathcal{P}_C$ was not satisfied.

Lemma 4.3: If there exists a violating path from an input to an output in any retimed circuit equivalent to $\mathcal{C}$, then the constraint for the corresponding path was not satisfied for $\mathcal{P}_C$.

Proof: By summing up the delay constraints along the violating path it is seen how the delay constraint for the corresponding path cannot be met in $\mathcal{P}_C$. Recall that a path is violating if the last segment is violating and all other segments are critical. Let the violating path have $j + 1$ segments. If $j = 0$, then $a + D + r \geq c + \delta$. The constraint on this path in $\mathcal{P}_C$ is that $(kc - r) - (a + (k - 1)c) > D$ or equivalently, $a + r + D < c$ (k is the label on the input). Thus, this constraint is violated. If $j > 0$, then the following inequalities are obtained from the fact that the path is violating.

$$a + D_o \geq c$$
$$D_i \geq c, \quad 1 < i < j$$
$$r + D_j \geq c + \delta.$$

From these we see that

$$a + \sum_{k=0}^{j} D_i + r \geq (j + 1)c. \tag{2}$$

The constraint on the same path in $\mathcal{P}_C$ is

$$((j + k)c - r) - (a + (k - 1)c) > \sum_{k=0}^{j} D_i.$$

This can be rewritten as

$$(j + 1)c > a + \sum_{k=0}^{j} D_i + r$$

which is not met as we can see from (2) above. □

Theorem 4.1: If $\mathcal{P}_C$ has a solution, then this can be retimed to give a solution of $\mathcal{P}_P$.

Proof: Follows from Lemmas 4.2 and 4.3. □

From Theorem 4.1 we see that solving $\mathcal{P}_C$ is sufficient in order to obtain a solution to $\mathcal{P}_P$ to within a gate delay. Now it is shown that any solution to $\mathcal{P}_P$ must be a retiming of a solution of $\mathcal{P}_C$.

Theorem 4.2: If there exists a solution to $\mathcal{P}_P$, then this can always be obtained by retiming some solution of $\mathcal{P}_C$.

Proof: For any path from an input to an output in the retimed circuit the following inequality must be satisfied for each segment: $\lambda_k > c$. Summing over all segments:

$$a + \sum_{k=0}^{j} D_i + r < (j + 1)c.$$

The constraint on this path in $\mathcal{P}_C$ is exactly this and is therefore satisfied. Since this is true for all paths, $\mathcal{C}_C$ for the solution of $\mathcal{P}_P$ is a solution of $\mathcal{P}_C$. The solution of $\mathcal{P}_P$ is then obtained by moving in the peripheral latches by retiming. □

Thus, for the pipelined problem to have a solution (within a gate delay) it is necessary and sufficient that the combinational problem has a solution. This is significant since it tells us that the problems are equivalent and, therefore, we need concentrate only on the relatively simpler combinational speedup problem.

TABLE I
EXPERIMENTAL RESULTS: PERFORMANCE OPTIMIZATION OF PIPELINED CIRCUITS

	Rysn-Ret			Ret-Rsyn			PR-Rsyn-Ret		
Name	Area	Latches	Cycle	Area	Latches	Cycle	Area	Latches	Cycle
ex1	804	27	14.4	518	28	15.4	758	25	14.4
ex2	500	25	16.6	542	22	15.4	571	42	11.0
ex3	292	9	12.6	265	14	14.0	371	22	12.4

V. EXPERIMENTAL RESULTS: PERFORMANCE OPTIMIZATION

5.1. Example Circuits and Experimental Procedure

Since there is no set of benchmark examples of pipelined circuits, we constructed a few simple examples for our experiments. The following three arithmetic circuits were designed for this purpose. Each of these is a two-stage pipelined circuit.

- **ex1** A two-stage adder that adds four 8-bit numbers (A, B, C, and D). The first stage computes the partial sums $A + B$ and $C + D$ and the second stage computes the final sum. Each adder is a ripple-carry adder.
- **ex2** A two-stage adder that adds two 16-bit numbers. The first stage computes the sum of the 8 least-significant bits and the second stage computes the final sum. Each adder is a ripple-carry adder.
- **ex3** This circuit computes the parity of the sum of two 8-bit numbers (A and B). During the first stage the sum is generated using a ripple-carry adder. In the next stage the parity of the sum is computed using a balanced parity tree.

Three design scenarios are evaluated, and the cycle time achieved by each is reported in Table I. The three scenarios explore different methods to obtain a faster version of the initial pipelined circuit and are as follows.

1) resynthesis followed by retiming (Rsyn-Ret),
2) retiming followed by resynthesis (Ret-Rsyn),
3) the approach proposed in Section IV, i.e., peripheral retiming followed by resynthesis followed by retiming (PR-Rysn-Ret).

For purposes of this experiment only one delay optimization routine is applied, the critical-path restructuring on a technology-independent network [12], as part of the resynthesis procedure. The delay through the circuit is measured using a two-input NAND gate representation of the circuit. Each gate contributes one unit of delay, and each fan-out contributes an additional delay (0.2 units in this experiment). The area of the combinational part is measured as the number of literals (gate input connections) in the same two-input NAND representation.

From the results in Table I we make the following observations:

1) The order of retiming and resynthesis operations impacts the value of cycle time that can be achieved. Neither order can be counted on to be the best for all circuits.

2) The cycle time obtained by the proposed method matches or is better than the best result that can be obtained by any combination of a single retiming and a single resynthesis step. This is what is expected theoretically, and it is gratifying that this is achieved in practice as well. It is clear from circuit *ex2* that the additional flexibility gained from looking at the maximal combinational logic sub-network obtained by peripheral retiming provides the optimization techniques greater freedom in restructuring the circuit to reduce the cycle time.

In the current experiments there is an increase in the number of latches, over the initial number of latches, when the proposed method is used. This is due to the fact that no attempt is made to minimize the latches during retiming and also due to the particular resynthesis technique used. The critical path restructuring increases the width of the circuit, and hence more latches are used.

VI. SUMMARY AND FUTURE WORK

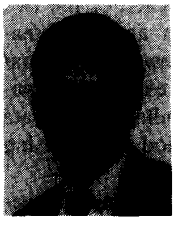
The paper presents a technique to resynthesize a pipeline circuit optimally to meet performance specifications. The proposed approach exploits combinational resynthesis as well as retiming techniques in a unified framework that guarantees the best possible circuit structure (under the constraint that the best combinational resynthesis techniques are used). The equivalence of the two problems, pipelined circuit optimization and combinational speedup, has been proved. This allows researchers/designers to focus only on the relatively easier combinational speedup problem. Preliminary results demonstrate the practical impact of the proposed method in increasing the performance of pipelined circuits.

In the future we would like to extend the method beyond pipelined circuits to general sequential circuits. In addition, several things need to be done to increase the practicality of this approach. We need to develop retiming procedures that consider fan-out loads. Stronger combinational resynthesis techniques need to be developed since they are key to improvement here.

REFERENCES

[1] K. A. Bartlett, G. Boriello, and S. Raju, "Timing optimization of multi-phase sequential logic," *IEEE Trans. Computer-Aided Design*, vol. 10, pp. 51-62, Jan. 1991.
[2] C. L. Berman, J. L. Carter, and K. F. Day, "The fanout problem:

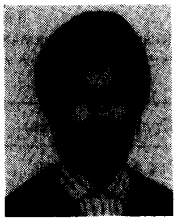
From theory to practice," in *Advanced Research in VLSI: Proc. 1989 Decennial Caltech Conf.*, pp. 69–99, Mar. 1989.
[3] G. De Micheli, "Performance-oriented synthesis of large-scale domino CMOS circuits," IEEE *Trans. Computer-Aided Design*, vol. 6, pp. 751–765, Sept. 1987.
[4] ——, "Synchronous logic synthesis: Algorithms for cycle-time minimization," *IEEE Trans. Computer-Aided Design*, vol. 10, pp. 63–73, Jan. 1991.
[5] E. Detjens, G. Gannot, R. Rudell, A. Sangiovanni-Vincentelli, and A. Wang, "Technology mapping in MIS," in *Proc. Int. Conf. Computer-Aided Design*, pp. 116–119, 1987.
[6] C. E. Leiserson, F. M. Rose, and J. B. Saxe, "Optimizing synchronous circuitry by retiming," in *Advanced Research in VLSI: Proc. Third Caltech Conf.*, pp. 23–36, 1983.
[7] C. E. Leiserson and J. B. Saxe, "Optimizing synchronous systems," *J. VLSI and Computer Syst.*, vol. 1, no. 1, pp. 41–67, Spring 1983.
[8] C. E. Leiserson and J. B. Saxe, "Retiming synchronous circuitry," *Algorithmica*, vol. 6, no. 1, pp. 5–36, 1991.
[9] S. Malik, "Combinational logic optimization techniques in sequential logic synthesis," Ph.D. dissertation, M90/115, Electronics Research Lab. Univ. of California, Berkeley, Nov. 1990.
[10] S. Malik, E. Sentovich, R. K. Brayton, and A. Sangiovanni-Vincentelli, "Retiming and resynthesis: Optimizating sequential networks with combinational techniques," *IEEE Trans. Computer-Aided Design*, vol. 10, pp. 74–84, Jan. 1991.
[11] P. G. Paulin and F. Poirot, "Logic decomposition algorithms for the timing optimization of multi-level logic," in *Proc. Int. Conf. Computer Design*, pp. 329–333, 1989.
[12] K. J. Singh, A. R. Wang, R. K. Brayton, and A. Sangiovanni-Vincentelli, "Timing optimization of combinational logic," in *Proc. Int. Conf. Computer-Aided Design*, pp. 282–285, 1988.
[13] H. Touati, C. Moon, R. K. Brayton, and A. Wang, "Performance-oriented technology mapping," in *Advanced Research in VLSI: Proc. Sixth MIT Conf.*, pp. 79–97, Apr. 1990.



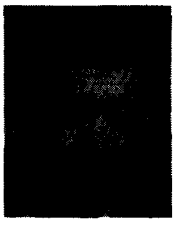
Sharad Malik received the B.Tech. degree in electrical engineering from the Indian Institute of Technology, New Delhi, India, in 1985 and the M.S. and Ph.D. degrees in computer science from the University of California, Berkeley, in 1987 and 1990, respectively.

Currently he is an Assistant Professor with the Department of Electrical Engineering, Princeton University, Princeton, NJ. His current research interests are in the synthesis and verification of digital systems.

Dr. Malik has been the recipient of the President of India's Gold Medal for academic excellence (1985) and an IBM Faculty Development Award (1991).



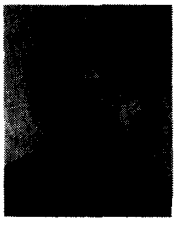
Kanwar Jit Singh received the B. Tech. degree in electrical engineering from the Indian Institute of Technology, Kanpur, India, in 1986. Currently he is a doctoral student and research assistant in electrical engineering at the University of California, Berkeley. His current research interests include delay modeling, performance optimization of combinational and synchronous circuits, timing analysis, and formal verification.



Rober 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 Science Department of the IBM T. J. Watson Research Center, Yorktown Heights, NY. In 1987, he joined the Department of Electrical Engineering and Computer Sciences at the University of California at Berkeley, where he is currently a Professor. He has authored or coauthored more than 150 technical papers, and is the coauthor of three books: *Logic Minimization Algorithms for VLSI Synthesis, Computer Aided Design: Sensitivity and Optimization,* and *Integrating Functional and Temporal Domains in Logic Design*. During the years 1965–1966 he was a visiting Professor at MIT, 1975–1976 at Imperial College, London, and 1985–1986 at the University of California at Berkeley. 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, combinational and sequential logic synthesis for area/performance/testability, and formal verification.

Dr. Brayton is a Fellow of the AAAS. He received the two Best Paper Awards from the IEEE Circuits and Systems Society (1971 and 1987) and one from HICSS'90. In addition, he has been the recipient of four IBM Outstanding Innovation Awards and two IBM patent awards.



Alberto Sangiovanni-Vincentelli (M'74–SM'81–F'83) received the Dr. Eng. degree from the Politecnico di Milano, Italy, in 1971.

From 1971 to 1977, he was with the Politecnico di Milano, Italy. In 1976, he joined the Department of Electrical Engineering and Computer Sciences, University of California, Berkeley, where he is presently a Professor. His research interests are in various aspects of computer-aided design an integrated circuits, with particular emphasis on VLSI, simulation, and optimization. He was an Associate Editor of the IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS, is currently an Associate Editor of the IEEE TRANSACTIONS ON COMPUTER AIDED DESIGN. In addition, he is a member of the Large-Scale Systems Committee of the IEEE Circuits and Systems Society and the Computer-Aided Network Design (CANDE) Committee. He was Executive Vice-President of the IEEE Circuits and Systems Society in 1983.

Dr. Sangiovanni-Vincentelli received the Distinguished Teaching Award from the University of California in 1981. At both the 1982 and the 1983 IEEE–ACM Design Automation Conference, he was given a Best Paper and a Best Presentation Award. In 1983, he received the Guillemin–Cauer Award for the best paper published in the IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS and the IEEE TRANSACTIONS ON COMPUTER-AIDED DESIGN in 1981–1982. He is a member of ACM and Eta Kappa Nu.