Incorporating Interconnect, Register, and Clock Distribution Delays into the Retiming Process

Tolga Soyata, *Member, IEEE,* Eby G. Friedman, *Senior Member, IEEE,* and James H. Mulligan, Jr.

***Abstract*— A retiming algorithm is presented which includes the effects of variable register, clock distribution, and interconnect delays. These delay components are incorporated into the retiming process by assigning register electrical characteristics (REC's) to each edge in the graph representation of a synchronous circuit. A matrix, called the sequential adjacency matrix (SAM), is presented that contains all path delays. Timing constraints for each data path are derived from this matrix. Vertex lags are assigned ranges rather than single values as in existing retiming algorithms. The approach used in the proposed algorithm is to initialize these ranges with unbounded values and to continuously tighten these ranges using localized timing constraints until an optimal solution is obtained. A branch and bound method is offered for the general retiming problem where the REC values are arbitrary. Certain monotonicity constraints can be placed on the REC values to permit the use of standard linear programming methods, thereby requiring significantly less computational time. These conditions and the feasibility of their application to practical circuits are presented. The algorithm is demonstrated on modified benchmark circuits and both increased clock frequencies and the elimination of all race conditions are observed.**

***Index Terms*— Clock distribution networks, clock scheduling, clock skew, clocking, interconnect delay, retiming.**

I. Introduction

RETIMING is a sequential optimization technique used to increase the clock frequency of synchronous circuits by relocating the registers in the circuit while maintaining the original function and latency of the system. Earlier retiming algorithms have assumed ideal conditions for the nonlogical portion of the data paths, specifically ignoring the temporal characteristics of the registers, the interconnect, and the clock distribution network. The authors are unaware of any retiming algorithms that consider variable register, clock distribution, and interconnect delays. Without including these delay components, existing retiming algorithms are not sufficiently accurate for their use in the development of practical high speed circuits. For this reason, clock distribution, variable register, and interconnect delays must be integrated into the retiming process in order to ensure that retiming becomes a practical and useful design methodology.

Manuscript received October 11, 1994; revised October 23, 1996. This work was supported by the National Science Foundation under Grant MIP-9208165. This paper was recommended by Associate Editor, M. Fujita.

T. Soyata and E. G. Friedman are with the Department of Electrical Engineering, University of Rochester, Rochester, NY 14627 USA.

J. H. Mulligan, Jr., deceased, was with the Department of Electrical and Computer Engineering, University of California, Irvine, CA 92717 USA.

Publisher Item Identifier S 0278-0070(97)01279-7.

Both register and interconnect delays are similar in magnitude to the delay of the logic elements. Also, variations in clock delay between widely separated registers may create clock skews which can drastically affect circuit operation. Undesirable clock skew can produce a net negative delay within a local data path. This implies the existence of a race condition, which must be avoided as a condition imposed on the retiming process.

In most retiming algorithms proposed to date, registers are assumed to have zero delay (e.g., [1], [2]) or equal delay (e.g., [3]). In [3], the setup (t_s) and hold (t_p) times are nonzero constant values, creating an effective clock period of $T_{PD}+t_s+t_p$, where T_{PD} is the worst case path delay of the synchronous circuit. Since constant register delays are assumed throughout the circuit, t_s+t_p is added to each individual local data path, biasing the clock period by this amount. However, this simple summation is not sufficiently accurate since each local data path typically has a different register delay.

Integrating clock skew into the retiming process was first proposed in [4] and [5]. The authors of this paper originally introduced the strategy of integrating clock skew and variable register delays into retiming by attaching electrical information describing the register to the edges of the graph representing the synchronous circuit [6]. These delay parameters are defined as register electrical characteristics (REC's) in this original work and are adhered to herein. Following this work, the integration of clock skew into retiming was discussed in [7]. In this paper, constraints are placed on the clock skew to permit the use of standard linear programming methods. Variable register and interconnect delays were not considered. In [8], a branch and bound algorithm is briefly introduced to solve the general retiming problem while considering variable nonzero clock skew and register and interconnect delays and is explained in greater detail in [9]. In general, there has been a growing interest in making retiming into a more practical and useful design methodology, as evidenced by [1]–[11].

The synchronous circuit optimization problem is approached in this paper as a two-step process:

1) optimization of the clock distribution network by buffer insertion and clock tree synthesis to meet a specified clock skew schedule [12]–[15];
2) optimization of the synchronous circuit via retiming given that the clock scheduling process has previously been performed to satisfy a specific set of clock skew specifications [6], [8], [9], [16].

Optimizing the clock distribution network followed by retiming may create a suboptimal result. This suboptimality

manifests itself in many practically applied algorithms, since optimality is sacrificed to prevent excessive computational times in existing algorithms. Ishii *et al.* published research results in simultaneous retiming and clock tuning [2]. Ishii *et. al.* report an $O(V^{11})$ algorithm to perform simultaneous retiming and clock tuning and comment that this problem is far too complicated to be practical unless optimality is sacrificed. They present a suboptimal $O(V^3(1/e)lg(1/e) + (VE + V^2lgV)lg(V/e))$ algorithm which calculates the retiming result with e% accuracy, where e is a user-selected error factor [2]. In this paper, the following methodology is assumed: clock skew scheduling followed by retiming.

A retiming algorithm is presented in this paper which incorporates variable register and interconnect delays and nonzero localized clock skew. Either rising edge or falling edge triggered D flip flops and a single phase clock are assumed throughout the synchronous digital circuit. To accomplish the integration of variable clock distribution, interconnect, and register delays into the retiming process, a path between logic elements is defined in this paper as the traversal from weighted edge to weighted edge, an edge being interpreted as a connection between logic elements containing zero, one, or more registers. With this definition, clock, register, and interconnect delays are assigned to each edge. Thus, as registers are shifted from edge to edge, different clock skews and register delays are considered in each of the local path delays. This permits both maximum clock periods and race conditions to be detected on a path-by-path basis. Estimates of register delays on zero weight edges (i.e., interconnections between logic elements that contain no registers) derived from the circuit layout are required in order to include the effects of variable register delays on the retimed circuit. This approach, therefore, initially requires approximate (or estimated) values of the register, clock distribution, and interconnect delays which can be replaced with more accurate values as the exploratory retiming process becomes better specified [6], [8], [9]. A simple strategy for estimating the clock delays is provided as an appendix.

The retiming algorithm *RETSAM* presented in this paper uses a branch and bound approach. Although *RETSAM* determines a retimed circuit that will operate at its maximum clock frequency, enhanced computational efficiency can be obtained by placing certain conditions related to the monotonicity of the path delays on the REC values. These monotonicity conditions permit the use of standard linear programming methods during the retiming process. These conditions and the feasibility of their application to practical circuits are presented in an appendix to this paper.

The paper is organized as follows. The background and definitions of important terms used throughout the paper are provided in Section II. In Section III, models of nonzero clock skew, variable register delay, and interconnect delay are presented. In Section IV, the sequential adjacency matrix (SAM) is introduced. Timing constraints, derived from the SAM, are described in Section V. The proposed retiming algorithm *RETSAM* is presented in Section VI. Monotonicity restrictions that may be placed on the REC's to permit the use of standard linear programming methods to perform retiming with the additional electrical delay information are described in Section VII. Results of applying the proposed algorithm *RETSAM* to benchmark circuits are presented in Section VIII and some conclusions are drawn in Section IX. A strategy for estimating the clock delays is presented in Appendix A. In Appendix B, the derivation of the monotonicity constraints is provided.

II. Background and Definitions

The absolute delay of the clock signal from the global clock source to a specific register (or memory element) is the **clock delay** and is denoted as T_{CD}. The difference between the clock delay of any two registers is the **clock skew** between these registers, denoted as T_{Skew}. The notion of **localized clock skew** and its application to increasing the clock frequency within pipelined systems was introduced by Friedman and Mulligan in [17]. They show that only the clock skew between **sequentially adjacent registers** (registers that receive information at successive clock intervals and are either directly connected or connected by logic elements) is significant in pipelined systems. A **local data path** is formed between two sequentially adjacent registers. The local data path with the greatest delay is the **critical data path**, whose delay defines the minimum clock period of the circuit.

The definition of sequential adjacency is extended in this paper to edges on a graph. **Sequentially adjacent edges** are those edges that are connected via a fully combinatorial path. The last register of the initial edge and the first register of the final edge are sequentially adjacent, thereby making the edges sequentially adjacent.

The clock skew T_{Skew} between two sequentially adjacent edges i and j is defined as

$$T_{\text{Skew}}(i,j) = T_{CD}(i) - T_{CD}(j). \tag{1}$$

If $T_{CD}(j) > T_{CD}(i)$, the clock skew between registers i and j is defined as being negative. **Negative clock skew** occurs if the initial clock signal leads the final clock signal of a local data path. If $T_{CD}(j) < T_{CD}(i)$, the clock skew between registers i and j is positive. **Positive clock skew** occurs if the initial clock signal lags the final clock signal of a local data path. In the case that $T_{CD}(j)$ equals $T_{CD}(i)$, i.e., the clock signal reaches the clock input of the two registers at precisely the same time, the clock skew is zero [18], [19].

Positive clock skew increases the path delay of a local data path, potentially making its local data path a critical path, whereas negative clock skew may improve circuit speed in critical paths [5], [18], [19]. However, it may also create negative path delays, resulting in **race conditions**. Race conditions are caused by *early-clocking*, i.e., clocking of registers before the relevant data is successfully latched. A race condition occurs if the skew is negative and greater in magnitude than the total local data path delay [5], [17]–[19]. Those paths with negative delay are called **short paths** [20]. Similarly, a **long path** designates those paths with a delay greater than the desired clock period of the circuit.

A synchronous circuit can be modeled by a graph composed of a vertex set V and an edge set E. $|V|$ and $|E|$ refer to

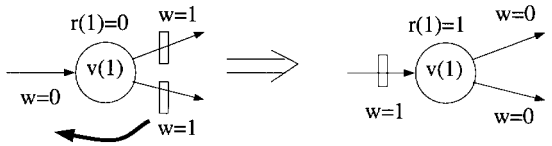


Fig. 1. The lag function changes the edge weights while preserving the circuit function. Increasing the lag of a vertex by one has the effect of increasing the weights of all edges by one connected in front of this vertex and decreasing the weights of all edges by one connected behind this vertex.

the cardinalities of these sets, i.e., the number of vertices and edges in the graph, respectively. Vertices denote logic elements and edges denote the connections between vertices. v_n and e_k represent vertex n and edge k, respectively. Every edge e_k connects two vertices. These two vertices are called the start vertex and the end vertex of e_k and are denoted as $e_k.start$ and $e_k.end$, respectively.

The logic element delay represented by v_n is $d(v_n)$ and is measured in **time units** (**tu**). The number of registers on an edge between two vertices is represented by the weight of the corresponding edge e_k and is denoted by $w(e_k)$. Edge-to-edge and vertex-to-vertex paths are represented by $e_i \rightsquigarrow e_j$ and $v_i \rightsquigarrow v_j$, respectively.

The **lag** of a vertex $v, r(v)$, is defined in [1] and adhered to herein. The vertex lag function $r(\ )$ plays a fundamental role throughout the entire retiming process and is therefore repeated here. The retiming process does not change the vertex delays $d(\ )$, however, the weights of the edges are changed according to the lags assigned to the vertices based on the following formula

$$w_r(e) = w(e) + r(e.end) - r(e.start) \tag{2}$$

where e is an edge, and $w(e)$ and $w_r(e)$ are the weight of edge e before and after retiming, respectively. Therefore, the retiming process can be thought of as determining $|V|$ integer vertex lags, $r(0) \cdots r(|V|-1)$, according to the retiming rules defined in [1]. An observation of (2) shows that instead of calculating $|E|$ edge weight values during retiming, $|V|$ vertex lags are calculated. Although the result is identical and yields a functionally equivalent circuit, the computational effort is significantly reduced since typically $|V|$ is much less than $|E|$, and the CPU time of these retiming algorithms depend polynomially on both of these unknowns, $|V|$ and $|E|$. The $r(\ )$ function is utilized since the edge weights do not change independently during the retiming process. The $r(\ )$ function represents the lags attached to the vertices denoting the relative edge weight adjustments, as shown in Fig. 1. Let $r(1)$ denote the lag of vertex 1. As shown in Fig. 1, changing the lag of the vertex from zero to one affects the weights of all of the edges connected to this vertex. Increasing the lag of a vertex by one has the effect of decreasing the weights of all the edges in front of the vertex by one and increasing the weight of all the edges behind the vertex by one. The retiming process consists of applying these vertex lag adjustments throughout the entire synchronous circuit to minimize the imbalance among all the path delays.

A W matrix, defined in [1], contains all vertex-to-vertex path weights. The elements of this matrix, $W(i,j)$, can be calculated as

$$W(i,j) = \min\{w(p)\colon p\colon v_i \rightsquigarrow v_j\}. \tag{3}$$

This matrix can be calculated using an all-pairs shortest path algorithm, such as the Floyd–Warshall algorithm [21]–[23]. Also, in this paper, a W_r matrix is defined as the W matrix after the retiming process has been applied to the circuit.

Each edge in the graph has a certain number of registers. Assume that e_k has N registers located on it. To distinguish between each register, the notation $e_k\colon R_n$ is introduced to denote the n^{th} register located on edge e_k.

An interconnect section between two nodes in a circuit has a delay T_{Int} with a distributed resistance-capacitance (*RC*) impedance with distributed resistance R_{Int} and distributed capacitance C_{Int}, respectively. Since each edge in the graph represents an interconnect line, every edge e_n can be thought of as being a distributed *RC* line with a delay of $T_{\text{Int}}(e_n)$ with a distributed resistance and capacitance of $R_{\text{Int}}(e_n)$ and $C_{\text{Int}}(e_n)$, respectively. In the event that a register separates the interconnect line, the interconnect delay of edge e_n is assumed to be separated into two values of $T_{\text{Int1}}(e_n)$ and $T_{\text{Int2}}(e_n)$, where T_{Int1} and T_{Int2} are the preregister and postregister delays, respectively. Three simplifying assumptions for the interconnect delays are the following.

1) $T_{\text{Int}}(e_n) = T_{\text{Int1}}(e_n) + T_{\text{Int2}}(e_n)$.
2) $T_{\text{Int1}}(e_n)$ and $T_{\text{Int2}}(e_n)$ are constant for $w(e_n) \geq 1$.
3) The interconnect delay is negligible between multiple registers on the same edge.

Increased generality and accuracy can be obtained by eliminating one or more of these assumptions at the expense of increasing the computational run time of the algorithm.

III. Register Electrical Characteristics (REC's)

In order to consider the effects of clock distribution, variable register, and interconnect delays, a number set, the REC, is assigned to each edge of the graph in the following form: $T_{CD}\colon T_{\text{Set-up}}/T_{C\rightarrow Q}$—$T_{\text{Int1}}/T_{\text{Int2}}$. T_{CD} is the clock delay from the global clock source to each register, $T_{\text{Set-up}}$ is the time required for the data at the input of a register to latch, $T_{C\rightarrow Q}$ is the time required for the data to appear at the output of the register upon arrival of the clock signal, and T_{Int} is the total interconnect delay along that edge and can be considered as being composed of two parts, T_{Int1} and T_{Int2}, if there is more than one register along that edge.

By attaching delay components to registers located on edges (connections between logic elements), the local path must be defined from edge-to-edge [6], [9] rather than vertex-to-vertex, as in existing retiming algorithms [1]. The original digital correlator introduced in [1] is depicted in Fig. 2. A modified version of this graph in which an REC is assigned to each edge is shown in Fig. 3. By assigning a clock delay to each edge (see Appendix A), the circuit is assumed to be partitioned into regions of similar clock delay, i.e., registers that are located on the same edge are physically located within the same clock delay region. Therefore, registers that end up on the same edge after retiming are assumed to have similar clock delay. Registers that move to different edges are assumed to have

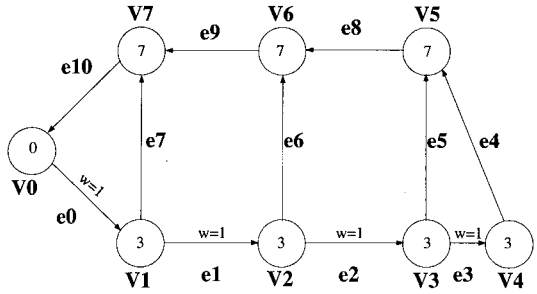


Fig. 2. Graph of the digital correlator in [1].

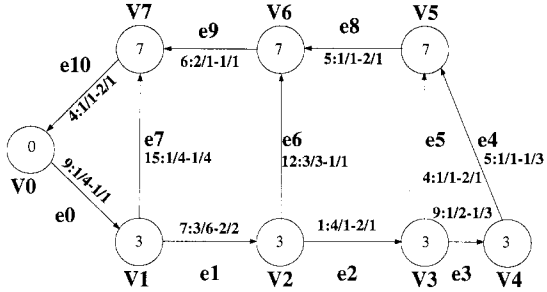


Fig. 3. Graph of the digital correlator [1] with added REC values.

the clock and register delays of the new edge. Since registers on different edges may be considered to have different clock and register related delays, moving a register from one edge to another edge during retiming will not only create different local data paths with different logic, register, and interconnect delays, but may also change the localized clock skew of the new local data paths.

The clock-to-Q delay $T_{C\rightarrow Q}(e_i)$ is edge dependent since each edge is connected to a different vertex, thereby changing the capacitive loading on the registers located on each edge. This variation occurs since each vertex represents a variety of possible logic elements, placing a different output load on the register driving the vertex. Hence, the same registers driving different vertex inputs will have different $T_{C\rightarrow Q}(e_i)$ delays. Furthermore, a variety of register types selected prior to the retiming process may be used at different locations within the circuit, due to the specific speed, power, and area tradeoffs peculiar to that portion of the circuit. $T_{\text{Set-up}}$ may also change for different register cell instances. Thus, $T_{\text{Set-up}}(e_i)$ can vary per edge. Therefore, selecting a specific register to satisfy a set of performance-based design requirements will change the value of $T_{\text{Set-up}}$ for each edge. A similar discussion is valid for $T_{C\rightarrow Q}(e_i)$. The varying loading exacerbates this delay variation, thereby requiring that variable register delays be considered during the retiming process.

The local data path delay $T_{PD}(i,j)$ from edges e_i to e_j is

$$T_{PD}(i,j) = T_{C\rightarrow Q}(i) + T_{\text{Int2}}(i) + T_{\text{Logic}}(i,j) + T_{\text{Int1}}(j) + T_{\text{Set-up}}(j) + T_{\text{Skew}}(i,j) \tag{4}$$

where $T_{\text{Logic}}(i,j)$ is the delay of the logic elements between e_i and e_j, including the interconnect delay of the zero weight edges along the path between these edges. If parallel paths exist, minimum and maximum local data path delays, $T_{PD_{\min}}$ and $T_{PD_{\max}}$, are defined. If $T_{PD_{\min}}(i,j) < 0$, a race condition between e_i and e_j exists since in this local data path the final register is clocked before the data signal arrives and is successfully latched.

If registers R_i and R_j are located on the same edge e_k and are sequentially adjacent, then, according to the definition of the REC's, the clock skew between R_i and R_j is zero from (1) since the clock delays of both registers are the same. This assumption is made since registers on the same edge would typically be physically close, and, therefore, the difference in clock delay to each register and the interconnect delay between these registers would be negligible. Furthermore, since no vertices (logic elements) exist between registers R_i and R_j, when both are on the same edge, the logic delay between the two registers is zero. Since all registers located on the same edge are defined to have the same timing characteristics (REC values), all sequentially adjacent registers located on the same edge have a similar internal path delay. A path composed of multiple registers on an edge could possibly be the critical worst case path of the overall circuit and its delay is defined as $T_{PD_{\text{Internal}}}(e_k)$, given by

$$T_{PD_{\text{Internal}}}(e_k) = T_{C\rightarrow Q}(e_k) + T_{\text{Set-up}}(e_k). \tag{5}$$

Although the likelihood that the delay of an internal path will be greater than the largest register-logic-register path delay is small, the worst-case delay of an internal path is considered in this paper for completeness. Certain circuits exist, such as a counter or shift register, in which this type of direct register-to-register path is common.

IV. Sequential Adjacency Matrix (SAM)

The sequential adjacency matrix (the SAM or the S matrix) is an $|E| \times |E|$ matrix whose element $S(i,j)$ is the path delay from e_i to e_j. The S matrix element, $S(i,j)$, is calculated from

$$S(i,j) = \max\{T_{PD}(i,j)\colon p\colon e_i \rightsquigarrow e_j \wedge w(p) = W(i,j)\}. \tag{6}$$

If parallel paths exist between any two edges, the S matrix is composed of two matrices, $S_{\min}$ and $S_{\max}$. Equations (7) and (8) are used to calculate the values of these two matrices. In order to reduce the number of matrices, a combined matrix, S', is used. $S'(i,j)$ contains $S_{\min}(i,j)$ if $S_{\min}(i,j)$ contains a zero or negative entry and contains $S_{\max}(i,j)$ if no zero or negative entry exists. The importance of $S_{\min}(i,j)$ is determined by whether a zero or negative entry exists, thereby denoting a race condition. If $S_{\min}(i,j)$ is completely positive, the maximum valued entries in $S_{\max}(i,j)$ limit the maximum speed of the circuit. Equation (9) is used to calculate the combined matrix, S'

$$S_{\min}(i,j) = \min\{T_{PD}(i,j)\colon p\colon e_i \rightsquigarrow e_j \wedge w(p) = W(i,j)\} \tag{7}$$

$$S_{\max}(i,j) = \max\{T_{PD}(i,j)\colon p\colon e_i \rightsquigarrow e_j \wedge w(p) = W(i,j)\} \tag{8}$$

$$S'(i,j) = \begin{cases} S_{\min}(i,j), & \text{if } S_{\min}(i,j) \le 0 \\ S_{\max}(i,j), & \text{if } S_{\min}(i,j) > 0. \end{cases} \tag{9}$$

Note that the S' matrix contains information for only those paths that can potentially cause the circuit to function

TABLE I
THE SAM FOR THE GRAPH OF FIG. 3. LIGHT SHADED ENTRIES REPRESENT SHORT PATHS, WHEREAS DARK SHADED ENTRIES REPRESENT LONG PATHS FOR $c = 24$ tu. UNSHADED ENTRIES DENOTE PERMISSIBLE PATHS

SAM		to										
		e0	e1	e2	e3	e4	e5	e6	e7	e8	e9	e10
from	e0	22	15	29	23	34	29	16	4	38	30	28
	e1	32	42	23	17	28	23	10	31	32	24	35
	e2	31	41	35	-1	10	5	42	30	14	23	34
	e3	43	53	67	61	14	67	54	42	26	35	46
	e4	31	41	55	49	60	55	42	30	14	23	34
	e5	28	38	52	46	57	52	39	27	11	20	31
	e6	28	38	52	46	57	52	39	27	61	20	31
	e7	26	36	90	44	55	90	37	25	59	51	29
	e8	19	29	43	37	48	43	30	18	52	11	22
	e9	11	21	35	29	40	35	22	10	44	36	14
	e10	-1	9	23	17	28	23	10	-2	32	24	22

improperly. Therefore, the zero and negative entries in the $S_{\min}$ matrix override the positive entries in the corresponding $S_{\max}$ matrix during the calculation of the S' matrix. This occurs since negative entries flag race conditions, and zero entries flag marginal race conditions and are not permitted to exist in the retimed circuit. To maintain a sufficient margin within the circuit, entries below a specific process dependent parameter k are not permitted. Paths with delays less than or equal to k tu may create race conditions due to statistical process variations within the integrated circuit and, therefore, are not permitted.

For the remainder of the paper, the notation for the combined matrix S' is denoted as S for simplicity. The S matrix of the graph of Fig. 3 is shown in Table I. The light shaded elements of the table indicate those paths with race conditions (negative values), and the dark shaded elements indicate those paths with a path delay greater than the desired clock period. In this example, a target clock period of 24 tu is assumed. Paths with zero delay are marginal race conditions that are not permitted and would appear as light shaded. The unshaded elements of the table indicate those paths that neither limit the maximum performance of the circuit nor create race conditions.

V. TIMING CONSTRAINTS

A branch and bound algorithm is presented in this paper in which unbounded values are initially assumed for the lag ranges. These lag ranges are tightened using timing constraints derived from the SAM. There are four different types of timing constraints: negative edge weight, long path, short path, and internal path. These different types of constraints are explained in greater detail in the following subsections.

A. Negative Edge Weight Constraints

As introduced in [1], a properly retimed graph contains no negative edge weights. Negative edge weights are permitted for peripheral edges in [24] in order to shift the registers to the periphery of a synchronous circuit. This approach permits combinatorial optimization to be performed on the circuitry placed between the peripheral edges. However, since the retiming algorithm described in this paper does not exploit this feature of **resynthesis**, negative edge weights are disallowed. Using (2), the negative edge weight constraint can be written as

$$w_r(e) \geq 0, \quad \forall e \in E. \tag{10}$$

B. Long Path Constraints

If a clock period c is desired, then all paths with a delay greater than c must be eliminated. Long paths are represented by entries in the S matrix that exceed a desired clock period c. In Table I, long paths for $c = 24$ tu are depicted using dark shaded elements. In order to eliminate these long paths, the two edges that create the long path are made nonsequentially adjacent.

Two registers are sequentially adjacent if there exists a zero weight path between them. According to this definition, in order to make two edges, e_i and e_j, nonsequentially adjacent, three approaches are possible: 1) the source or 2) the destination edges can be made zero weight, i.e., all registers can be removed from these edges, or 3) one or more registers can be placed within each zero weight path between the source and destination registers. The first two conditions exist since by eliminating the initial and/or final register of a local data path, a longer path is created which may have a smaller delay (due to negative clock skew). Using the definitions for w_r and W_r described in Section II, these three conditions can be written in terms of path and edge weights as follows:

$$w_r(e_i) = 0 \tag{11}$$

$$w_r(e_j) = 0 \tag{12}$$

$$W_r(e_i.end, e_j.start) > 0. \tag{13}$$

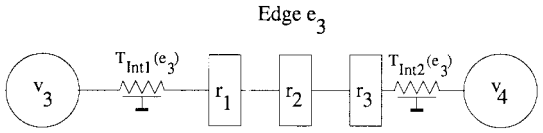


Fig. 4. The internal path delay between registers located on the same edge, $T_{PD_{\text{Internal}}}$, is equal due to the definition of the REC's. This example demonstrates the case where $w(e_3) = 3$.

If (11) or (12) is satisfied, then no registers exist on edge i or j, respectively, and, therefore, all local data paths between edges i and j are eliminated. If (13) is satisfied, all possible paths between edges i and j have a weight of at least one. This violates the definition of sequential adjacency, i.e., no paths exist with a zero weight between these two edges. Intuitively, it is stated in (11)–(13) that either the initial or the final edge does not have any register located on it, or there is at least one register along every path between these two edges.

C. Short Path Constraints

Short paths appear as zero or negative entries in the S matrix. $S(i,j) \leq 0$ indicates a short path originating at e_i and terminating at e_j. If e_i and e_j form a short path, then the initial and final registers of this path must be made nonsequentially adjacent. Equations (11)–(13) are used to eliminate any catastrophic short paths (or race conditions).

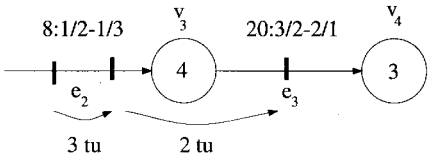
D. Internal Path Constraints

Internal long paths are created between two sequentially adjacent registers on the same edge when the edge weight is greater than one, and the internal path delay is greater than a specified clock period c. Internal long path constraints can be formulated using (5) as

$$w_r(e_i) \leq 1, \quad \forall i\colon T_{PD_{\text{internal}}}(e_i) > c \tag{14}$$

which suggests that if the internal path delay of an edge is greater than the desired clock period, the weight of that specific edge must be either zero or one to prevent internal long paths.

An edge with a weight of three is depicted in Fig. 4. More precisely, edge e_3 of Fig. 2 which connects vertices v_3 and v_4 is shown under the assumption, $w(e_3) = 3$. In Fig. 4, the internal path delay on edge e_3 (the delay between register pairs r_1, r_2 and r_2, r_3) is constant since multiple registers on the same edge are assumed to have the same delay, and the interconnect delay between internal registers is assumed to be negligible (see assumption 3 in Section II). Since the electrical characteristics of the vertices and registers do not change, the internal path delay is calculated from (5) only once before the retiming process is applied. This procedure ensures that before the retiming process starts, unnecessary internal long paths due to excessive internal path delays are not created. An example graph in which the internal path delay of edge e_2 exceeds the path delay between edges e_2 and e_3, thereby causing an internal long path, is shown in Fig. 5. It can be observed from Fig. 5 that the internal long path delay exceeds the path delay between e_2 and e_3 since the negative clock skew between these registers decreases the path delay, lowering $T_{PD}(e_2, e_3)$ below the internal path delay.



$$T_{PD_{Internal}}(e_2) > T_{PD}(e_2, e_3)$$

Fig. 5. An example graph in which the internal path delay on edge e_2 exceeds the path delay between e_2 and e_3. This graph exemplifies the importance of considering internal long paths before the retiming process is initiated.

Note that **internal short paths** are not possible since the clock skew between any two registers on the same edge cannot be negative (the clock skew must be zero). Therefore, internal short paths are not considered in this paper.

E. Constraints Due to Vertex Lags

Constraints (10), (11), (12), (13), and (14) are written in terms of edge weights. These constraints can be rewritten as (15), (16), (17), (18), and (19), respectively, to reduce the number of necessary operations.

$$r(e.start) - r(e.end) \leq w(e), \quad \forall e \in E \tag{15}$$

$$r(e_i.start) - r(e_i.end) = w(e_i) \tag{16}$$

$$r(e_j.start) - r(e_j.end) = w(e_j) \tag{17}$$

$$r(e_i.end) - r(e_j.start) \leq W(i,j) - 1 \tag{18}$$

$$r(e_i.start) - r(e_i.end) \geq w(e_i) - 1 \quad \forall i\colon T_{PD_{\text{internal}}}(e_i) > c. \tag{19}$$

In order to provide some intuition to (15), (16), (17), (18), and (19), note that, given two vertices u and v, the value $r(u) - r(v)$ can be thought of as "the number of registers taken out of the path p: $u \rightsquigarrow v$." Given this interpretation, it is implied in (15) that "the number of registers taken from an edge e cannot be greater than the original weight of the edge," i.e., none of the edge weights can be negative. In a similar manner, it is stated in (16) and (17) that "the number of registers taken from edge e_i and e_j, respectively, must be equal to the original weight of this edge," implicitly stating that this edge should be made zero weight. In (18) it is stated that "the registers taken from the path p: $e_i \rightsquigarrow e_j$ must be less than the original weight of this path minus one," implicitly stating that at least one register should be left along any path between registers e_i and e_j, thereby making this path nonsequentially adjacent. Finally, in (19) it is implied that either zero or one register should be left on an edge e that contains an internal long (worst case) path.

VI. Retiming Algorithm

In this section three algorithms are introduced: 1) Algorithm *RETSAM* to perform retiming of synchronous circuits, 2) Algorithm *CHECKCP* to check the feasibility of a specific clock period, and 3) Algorithm *SOLVELAGS* to determine the vertex lags based on a branch and bound method. These three algorithms are explained in the following subsections.

1. $CP_{min} = 0$
2. CP_{max} = clock period of the original graph
3. Calculate SAM
4. If the original graph has race conditions $CP_{max} = \max\{S(i,j),\ \forall i,j\}$,
5. Choose $CP_{target} = \lfloor \frac{CP_{max}+CP_{min}}{2} \rfloor$,
6. Check for feasibility of $c = CP_{target}$ using algorithm *CHECKCP*
7. If set of inequalities can be successfully solved, then $CP_{max} = CP_{target}$
8. If not, then $CP_{min} = CP_{target}$
9. Continue this process until $CP_{min} = CP_{max}$

Fig. 6. Pseudocode for *RETSAM*.

1. $r[0] = [0 \ldots 0]$
2. $r[k] = [-\infty \ldots +\infty], k = 1, \ldots, E-1$
3. Create a constraint list L for clock period c
4. Adjust lags to satisfy all constraints using algorithm *SOLVELAGS*
5. If all lags are fixed and all constraints are unsatisfied

   → Clock period is not feasible
6. If all constraints are satisfied

   → Clock period is feasible
7. If c is feasible and all lags are not fixed

   → Use lower bounds of the unfixed lags

Fig. 7. Pseudocode for clock period feasibility test, CHECKCP.

A. RETSAM: Retiming Algorithm for Synchronous Circuits with Attached Electrical Information

Retiming a synchronous circuit is achieved by performing a binary search of all possible clock periods on a specific circuit graph. The pseudocode of the retiming algorithm is shown in Fig. 6. The lower and upper bounds of the binary search are $CP_{\min}$ and $CP_{\max}$, respectively. Initially the lower bound is zero (Step 1). If the original graph does not contain any race conditions, the critical path delay of the original graph defines the upper bound of the binary search (Step 2). The SAM is calculated in Step 3 and used throughout the algorithm. If the original graph contains one or more race conditions, the maximum value in the SAM is used as the upper bound (Step 4). During the binary search, a specific clock period, CP_{target}, is checked for feasibility using algorithm *CHECKCP* (Steps 5 and 6). Depending on whether a solution exists (Step 7) or not (Step 8), the lower and upper search bounds are adjusted, and the binary search continues until the minimum clock period is determined (Step 9). An approximate solution can be obtained for the minimum clock period if *RETSAM* is terminated once the binary search bounds become sufficiently tight. This may significantly reduce the run time requirement of the algorithm, since target clock periods close to the minimum clock period may require excessive computational time.

B. CHECKCP: Clock Period Feasibility Check

A feasibility check for a specific clock period CP_{target} is achieved by solving the set of nonlinear inequalities for the vertex lag ranges. If all the constraints are satisfied for every path in the graph, the clock period is considered feasible. Pseudocode for the algorithm that determines the feasibility of a clock period is shown in Fig. 7. Lag ranges are stored in an array called $r[\,]$. The timing constraints are derived from the SAM.

The most important step in *CHECKCP* is solving for the vertex lags $r(\,)$ using Algorithm *SOLVELAGS*. The objective of the retiming algorithm is to yield a set of vertex lags that satisfy (15)–(19). To achieve this objective, the vertex lag ranges are initialized with unbounded values $[-\infty \cdots \infty]$. Timing constraints are continuously applied to these vertex lags in order to tighten the ranges until eventually all the constraints are satisfied. Once the vertex lags are each defined, these lag values are used to determine the edge weights of the retimed graph according to (2).

C. SOLVELAGS: Determination of the Vertex Lags Using a Branch and Bound Approach

The following types of equalities and inequalities are created from the aforementioned timing constraints:

$$r(v_a) - r(v_b) = k \tag{20}$$
$$r(v_a) - r(v_b) \le k \tag{21}$$
$$r(v_a) - r(v_b) = k_1 \quad \text{or} \quad r(v_c) - r(v_d) = k_2 \tag{22}$$
$$r(v_a) - r(v_b) = k_1 \quad \text{or} \quad r(v_c) - r(v_d) \le k_2 \tag{23}$$
$$r(v_a) - r(v_b) = k_1 \quad \text{or} \quad r(v_c) - r(v_d) = k_2 \quad \text{or} \quad r(v_e) - r(v_f) \le k_3 \tag{24}$$

where $r(\,)$ are vertex lags and k_n are constants. The *or* statements that appear in (22), (23), and (24) prohibit the use of standard linear programming methods [21] and necessitate the use of branch and bound techniques for the general unconstrained retiming problem. However, it is shown in Section VII that a polynomial-time suboptimal solution is feasible when the path delays are constrained to monotonically increasing delay values.

Note the existence of *multiple choices* in each inequality in (22), (23), and (24) in the form of $s1$ *or* $s2$ *or* $s3$, where $s1, s2$, and $s3$ are different choices. $s3 = nil$ (nonexistant) implies the form shown in (22) and (23), whereas $s3 = nil$ and $s2 = nil$ imply the form shown in (20) and (21). Notationally, s1 must always exist, and $s2 = nil$ and $s3 \neq nil$ is not permitted. Therefore, (24) is characterized by $s3 \neq nil$. Note that it is possible to *reduce* the complexity of a multiple choice inequality by either eliminating $s2$ or $s3$. Thus, an inequality originally in the form of (24) can be converted to the form of (22), thereby reducing its complexity.

To gain insight into how these multiple choice inequalities are created, consider retiming the graph of Fig. 3, for which the SAM is shown in Table I. To achieve a clock period of $c = 24$ tu, the dark shaded and light shaded paths must be avoided, since they represent long paths for $c = 24$ tu and short paths, respectively. To avoid, for example, the path p: $e_3 \rightsquigarrow e_0$, there exists three possible choices, derived from (11), (12), and (13), resulting in the multiple choice inequality

$$r(3) - r(4) = 1 \quad \text{or} \quad r(0) - r(1) = 1 \quad \text{or} \quad r(4) - r(0) \le -1 \tag{25}$$

which states that to eliminate the path starting at e_3 and terminating at e_0, either e_3 or e_0 must be zero weight, thereby making the path p: $e_3 \rightsquigarrow e_0$ nonexistent, or at least one register

```
1.  Let L be the constraint list.
2.  while L ≠ ∅
3.    Let l = next constraint in L in the form of s1 or s2 or s3, where s1, s2, or s3 is one of
      the timing constraints. Note that s2 and/or s3 may be nil (nil is always non-satisfiable,
      thereby, having no effect on l).
4.    if s3 = nil goto 12.
5.    Constrain vertex lags to satisfy s3. Iterate if necessary.
6.    if s3 is non-satisfiable with the current vertex lag list
7.      l = l − s3 (delete condition s3).
8.    else
9.      L = L − l (delete the entire constraint)
10.     goto 18.
11.   endif.
12.   if s2 = nil goto 14.
13.   repeat steps 5 through 11 for s2.
14.   Constrain vertex lags to satisfy s1. Iterate if necessary.
15.   if s1 is non-satisfiable using the current vertex lag list
16.     exit. no solution exists.
17.   else L = L − l (delete the entire constraint).
18. endwhile.
19. Use the vertex list as the solution
```

Fig. 8. Pseudocode for the branch and bound algorithm *SOLVELAGS* that calculates the vertex lags.

must be placed between the initial and terminating vertices of the path p.

The pseudocode of the branch and bound algorithm *SOLVELAGS* that calculates the vertex lags is shown in Fig. 8. A list L is maintained to store the timing constraints derived from the SAM (Step 1) which are individually eliminated until no more constraints remain unevaluated (Steps 2 and 18). Within this loop, each constraint l is evaluated separately (Step 3) to determine if $s3$ (Step 4) or $s2$ (Step 12) is nonexistant.

If $s3$ can be eliminated (i.e., $s3$ can never be satisfied using the current vertex lag list), the complexity of the constraint l can be reduced (Steps 5–7). Alternatively, if $s3$ is satisfied using the current vertex list without further tightening the boundaries of the vertex lag list, the constraint l can be eliminated (Steps 8–11). The same operations are performed on condition $s2$ between Steps 12 and 13.

After $s3$ and $s2$ are evaluated, the lags are adjusted to satisfy constraint $s1$ (Step 14). If $s1$ cannot be satisfied, a vertex lag set does not exist that satisfies all constraints (Steps 15 and 16), since $s1$ is the last possible solution in the *or* chain. If a vertex lag set can be found that satisfies $s1$, the vertex lag that satisfies $s1$ is used (Step 17). After L is completely evaluated and the entire list of constraints is satisfied, a solution exists and the current status of the vertex list is the set of final vertex lags (Step 19).

The solution method for determining the vertex lags of the graph shown in Fig. 3 is exemplified in Table II. The target minimum clock period in this example is 24 tu. To solve for a set of vertex lags that provide a proper retiming, an inequality similar to (25) is written for each short or long path shown in Table I. In this algorithm, the unbounded value $[-\infty \cdots \infty]$ is initially assigned to each vertex lag range. Only one vertex lag $(r(0)$ for simplicity) is initialized to zero and the other lags are calculated relative to $r(0)$. Bounds of the vertex lag ranges are continuously tightened to determine a set of vertex lag ranges that satisfies all of the constraints.

In Table II, the vertex lag nonnegativity constraint from (15) is applied to each vertex to further tighten the vertex lag ranges (shown in the first three rows). When the constraint from (15) cannot be used to further tighten the vertex lag ranges, the long path constraints from (16), (17), and (18) are used (see row 4). Short paths are also eliminated using (16), (17), and (18). Each time the bounds are tightened by applying long (or short) path constraints, (15) is applied to the new set and the neighboring vertex lag ranges to ensure nonnegative edge weights on each edge. The algorithm may reach a point where the application of the constraints can no longer tighten the bounds (the dark shaded row). Once this occurs, all possible values for each vertex lag are tested. On the first dark shaded row in Table II, there are two unfixed lags with cardinalities two and three, respectively. Therefore, $2 * 3 = 6$ possible solutions exist and must be evaluated. If a solution is reached, the algorithm is terminated and the resulting vertex lag ranges are used to determine the edge weights of the retimed graph. If all possible solutions are considered and a set of vertex lag ranges cannot be determined that satisfy all constraints, a solution for that specific clock period does not exist.

VII. Path Delay Monotonicity Constraints

The algorithm *RETSAM* is capable of including arbitrary register properties, including clock delays. In the event, however, that the design freedom permits consideration of the REC delay values as part of the design process, certain constraints can be placed on the REC values which will permit the use of standard linear programming techniques, such as the Bellman–Ford method [21], for solving for the vertex lags. This process yields more computationally efficient results. In the following subsections, the path delay monotonicity constraints that must be applied to the REC values to permit computationally efficient retiming are introduced, design issues relating to the clock distribution network with respect to satisfying the monotonicity constraints are described, and

TABLE II
EXAMPLE SOLUTION FOR $c = 24$. A SINGLE VALUE IS SHOWN FOR EQUAL, LOWER, AND UPPER BOUNDS

Constraint	Type	r(0)	r(1)	r(2)	r(3)	r(4)	r(5)	r(6)	r(7)
		0	−∞..∞	−∞..∞	−∞..∞	−∞..∞	−∞..∞	−∞..∞	−∞..∞
r(0)-r(1)≤1	Negativity on e_0	0	−1..∞	−∞..∞	−∞..∞	−∞..∞	−∞..∞	−∞..∞	−∞..∞
r(1)-r(7)≤0	Negativity on e_7	0	−1..∞	−∞..∞	−∞..∞	−∞..∞	−∞..∞	−∞..∞	−1..∞
Negativity on e_1, e_2, e_3, e_4, e_5, e_6, e_7, e_8, e_9		0	−1..0	−2..0	−3..1	−4..0	−3..0	−2..0	−1..0
r(1)-r(7)=0 or r(7)-r(1)≤0	Long path: $e_7 \rightarrow e_7$	0	-1	−2..0	−3..1	−4..0	−3..0	−2..0	-1
r(6)-r(7)≤0	Negativity on e_9	0	-1	−2..0	−3..1	−4..0	−3..0	−2.. − 1	-1
r(5)-r(6)≤0	Negativity on e_8	0	-1	−2..0	−3..1	−4..0	−3.. − 1	−2.. − 1	-1
r(4)-r(5)≤0	Negativity on e_4	0	-1	−2..0	−3..1	−4.. − 1	−3.. − 1	−2.. − 1	-1
r(3)-r(5)≤0	Negativity on e_5	0	-1	−2..0	−3..−1	−4.. − 1	−3.. − 1	−2.. − 1	-1
r(2)-r(6)≤0	Negativity on e_6	0	-1	−2.. − 1	−3..−1	−4.. − 1	−3.. − 1	−2.. − 1	-1
r(2)-r(6)=0 or r(7)-r(0)=0 or r(6)-r(7)≤-1	Long path: $e_6 \rightarrow e_{10}$	0	-1	−2.. − 1	−3..−1	−4.. − 1	−3.. − 1	-2	-1
r(5)-r(6)≤0	Negativity on e_8	0	-1	−2.. − 1	−3..−1	−4.. − 1	−3.. − 2	-2	-1
r(4)-r(5)≤0	Negativity on e_4	0	-1	−2.. − 1	−3..−1	−4.. − 2	−3.. − 2	-2	-1
r(3)-r(5)≤0	Negativity on e_5	0	-1	−2.. − 1	−3..−2	−4.. − 2	−3.. − 2	-2	-1
r(2)-r(6)≤0	Negativity on e_6	0	-1	-2	−3..−2	−4.. − 2	−3.. − 2	-2	-1
Iteration #1: choose r(5)=-2		0	-1	-2	−3..−2	−4.. − 2	-2	-2	-1
Iteration #1: choose r(3)=-3		0	-1	-2	-3	−4.. − 2	-2	-2	-1
r(2)-r(3)=1 or r(3)-r(4)=1	Short path: $e_2 \rightarrow e_3$	Condition 1 satisfied							
r(3)-r(4)≤1	Negativity on e_3	0	-1	-2	-3	-2	-2	-2	-1
r(3)-r(4)=1 or r(6)-r(7)=0 or r(4)-r(6)≤-1	Long path: $e_3 \rightarrow e_9$	Constraints are not satisfiable							
Iteration #2: choose r(5)=-3		0	-1	-2	−3..−2	−4.. − 2	-3	-2	-1
⋮									

the feasibility of applying these monotonicity constraints to practical circuits is discussed.

A. Path Delay Monotonicity Constraints

In practical integrated circuits, variations in clock delay between widely separated registers may create clock skews which can drastically affect circuit operation. An observation of (4) is that arbitrary clock skews (in particular, negative clock skews) may cause longer data paths (paths with more edges and vertices) to have less delay than shorter data paths (paths with less edges and vertices). Therefore, unless constraints are placed on the possible clock delays, the data path delays are arbitrary and can quite possibly be negative. This decrease in path delay can occur either due to negative clock skew or to the delay components of the newly placed register being less than the delay components of the original register. Thus, a subpath p_1 of a longer path p (composed of added edges and vertices) may have a delay greater than path p. An example graph in which this occurs is depicted in Fig. 9. In this graph, the subpath p_1 has a greater delay than path p. The primary cause is due to the effect of negative clock skew which effectively subtracts delay from the local data path p, thereby causing the subpath p_1 to have greater delay (or the longer path p to have less delay). In the specific example shown in Fig. 9, subpath p_1 is 1 tu greater than path p. This 1 tu difference results

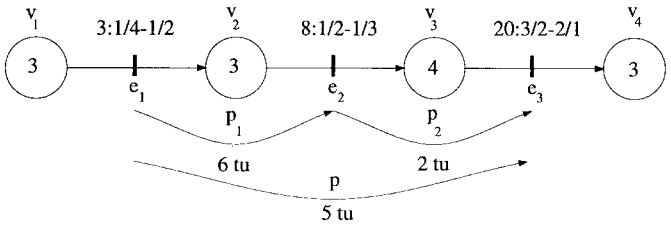


Fig. 9. An example graph in which the path delays do not monotonically increase. The subpath p_1 has a delay greater than its original path p. The cause of this nonmonotonic behavior is due to the negative clock skew between edges e_1 and e_2.

1. Calculate the negative clock skew tolerance [$\tau(a,b)$] for each local data path $e_a \rightsquigarrow e_b$ in the synchronous circuit
2. Calculate the positive clock skew tolerance [$\mathrm{T}(a,b)$] for each local data path $e_a \rightsquigarrow e_b$ in the synchronous circuit
3. Write the inequalities for each local data path using (30) and (31)
4. Solve the inequalities with the Bellman-Ford method
5. Use the resulting clock delays to design the clock distribution network

Fig. 10. Pseudocode version of algorithm for designing the clock distribution network.

from the negative clock skew between edges e_1 and e_2, i.e., $T_{\mathrm{Skew}}(e_1, e_2) = 3 - 8 = -5$ tu.

When a subpath of a larger path has a greater delay, there are three choices for removing that path. These choices are the following.

1) Place a register between the initial and the terminal edges, since a shorter path may have a smaller delay (or a short path may have a larger delay).
2) Remove the initial edge so that the path becomes longer (more edges and vertices). This longer path may have a smaller delay.
3) Remove the terminal edge so that the path becomes longer. This longer path may have a smaller delay.

Conditions 2) and 3) are required since the data path delays are completely arbitrary, and any of these conditions may possibly remove the undesirable path. Since these three conditions are used in *RETSAM*, standard linear programming techniques are not possible due to the boolean *or* operation, thereby resulting in the multiple-choice inequalities in (20)–(24).

A strategy to improve the time efficiency of the retiming algorithm is as follows: If certain temporal constraints are placed on the clock delays, it is possible to guarantee that a subpath p_1 of a larger path p will always have a larger delay, thereby removing the aforementioned conditions 2) and 3). This simplification permits using the standard Bellman–Ford method, since by removing conditions 2) and 3), the remaining inequalities are linear in the form of $x_i - x_j \leq a_{ij}$, since no boolean *or* operation is being performed.

Assume a path with two edges e_a and e_b, respectively, and a vertex v_a between these edges. The following condition to ensure monotonically increasing path delays is as follows:

$$T_{PD}(a,b) \geq \max\{T_{\mathrm{REG}}(a), T_{\mathrm{REG}}(b)\} \tag{26}$$

where $T_{\mathrm{REG}}(a)$ and $T_{\mathrm{REG}}(b)$ are the sum of the setup and clock-to-Q delays of the registers located on edges e_a and e_b, respectively. Details of the derivation of (26) can be found in Appendix B.

Equation (26) guarantees monotonically increasing delays for each edge-to-edge local data path. If (26) is satisfied and all path delays increase monotonically, standard linear programming methods can be applied when retiming a graph, thereby dramatically improving the computational efficiency of the retiming process. Relationship (26) does not permit race conditions since race conditions may create subpaths with delays larger than the original paths. Therefore, race conditions must be eliminated in advance to permit the use of the Bellman–Ford method. Therefore, this strategy does not verify the existence of race conditions but instead assumes that all race conditions have been eliminated *a priori*. Given that (26) is satisfied for each path in the synchronous circuit, inequalities for longer paths can be written and solved using the Bellman–Ford method. Equations (27) and (28) must be satisfied to ensure that a proper retiming with REC's has been accomplished. These conditions are similar to those derived in [1]

$$r(u) - r(v) \leq w(e), \quad \forall e\colon u \rightarrow v \tag{27}$$

$$r(u) - r(v) \leq W(u,v) - 1, \quad \forall i,j \in E\colon S(i,j) > c, \quad u = e_i.end, v = e_j.start. \tag{28}$$

B. Designing the Clock Distribution Network

In a practical integrated circuit, clock delays to each individual register may vary significantly due to the layout characteristics of the clock distribution network, creating localized clock skew between sequentially adjacent registers. These initial clock delay values can be changed by redesigning the clock distribution network, for example, by inserting buffers into certain clock paths. By applying this type of methodology, a clock distribution network can be designed which maintains monotonically increasing path delays, thereby satisfying (29) and (30)

$$T_{CD}(b) - T_{CD}(a) \leq \tau(a,b) \tag{29}$$

$$T_{CD}(a) - T_{CD}(b) \leq T(a,b). \tag{30}$$

Equations (29) and (30) represent the minimum and maximum clock skew that each individual local data path may have without causing improper circuit operation. As described in Appendix B, $\tau(a,b)$ and $T(a,b)$ are the negative clock skew tolerance and the positive clock skew tolerance of path $e_a \rightsquigarrow e_b$, respectively.

Careful observation of (29) and (30) will show that these expressions represent a family of inequalities which can be solved using the Bellman–Ford method [21]. Therefore, a clock distribution network can be systematically designed with this methodology, as shown by the pseudocode algorithm presented in Fig. 10. The process described in Step 5, designing the clock distribution network from the individual clock delays, is discussed in greater detail in [12]–[15].

In the event that no solution for this set of inequalities exists, a clock distribution network design is not feasible. If this occurs, these monotonicity conditions cannot be satisfied, and the algorithm *RETSAM* can be used.

A key aspect of the results of this research is the close interaction that exists between the design of the clock distribution network and the computational efficiency of the retiming process. It is shown herein that if the clock distribution network is poorly designed, the retiming process

may be greatly degraded. Alternatively, a well designed clock distribution network may significantly enhance the efficiency of the retiming process.

C. Feasibility Check for the Clock Distribution Network

To verify whether the conditions imposed on the clock delays are feasible in practical circuits, typical values for the REC's are used to exemplify the design process. The original digital correlator presented by Leiserson-Saxe and shown in Fig. 2 is used as an example circuit. The logic elements on the vertices v_1, v_2, v_3, and v_4 are comparators and are modeled as XNOR gates with a nominal delay value of 3.5 ns. The logic elements on vertices v_5, v_6, and v_7 are full adders with a nominal delay value of 4.0 ns. Typical register setup and clock-to-Q times of 4.0 ns and 3.0 ns are used. These temporal values are derived from industrial-based standard cell libraries. Initially, zero clock skew is assumed to predict the negative clock skew tolerance of a path.

The parameters are provided below for an arbitrary path p: $e_0 \rightsquigarrow e_1$ in Fig. 2. Interconnect delays $T_{\text{Int}1}$ and $T_{\text{Int}2}$ are assumed to each be 1.4 ns, approximately 20% of the register delays. In this case, for edges e_0 and e_1, the register delays are

$$T_{\text{REG}}(e_0) = T_{\text{REG}}(e_1) = 4.0 + 3.0 = 7.0 \text{ ns}$$

and, therefore

$$\max\{T_{\text{REG}}(e_0), T_{\text{REG}}(e_1)\} = 7.0 \text{ ns}.$$

Assuming zero clock skew, i.e., $T_{\text{Skew}}(e_0, e_1) = 0$, the path delays are

$$\begin{aligned} T_{PD}(e_0, e_1) &= T_{C\rightarrow Q}(e_0) + T_{\text{Int}2}(e_0) + d(v_1) + T_{\text{Int}1}(e_1) \\ &\quad + T_{\text{Set-up}}(e_1) + T_{\text{Skew}}(e_0, e_1) \\ &= 3 + 1.4 + 3.5 + 1.4 + 4 + T_{\text{Skew}}(e_0, e_1) \\ &= 13.3 \text{ ns}. \end{aligned}$$

According to (26)

$$T_{PD}(e_0, e_1) \geq \max\{T_{\text{REG}}(e_0), T_{\text{REG}}(e_1)\} \qquad (31)$$

which is satisfied since

$$13.3 \text{ ns} \geq 7.0 \text{ ns}.$$

It can be shown that these conditions are satisfied for all paths in the digital correlator circuit shown in Fig. 2. Therefore, as long as the clock skew is zero, (26) is satisfied, and the digital correlator has monotonically increasing path delays throughout the entire circuit. Now let $T_{\text{Skew}}(e_0, e_1) = T_{CD}(e_0) - T_{CD}(e_1)$ in Fig. 2 be negative. Positive clock skew is not considered here since it does not affect the monotonicity constraint. Applying negative clock skew, the inequalities become

$$T_{\text{Skew}}(e_0, e_1) + 13.3 \text{ ns} \geq 7 \text{ ns}$$

or

$$T_{\text{Skew}} \geq -6.3 \text{ ns}.$$

Thus, the negative clock skew tolerance of the local data path is 6.3 ns, i.e., negative clock skew is permitted, however, it cannot exceed a magnitude of 6.3 ns. A methodology for designing clock distribution networks based on nonzero localized clock skew is described in greater detail in [12]–[15].

VIII. Experimental Results

The retiming algorithm *RETSAM* is implemented in C on a SUN 4 workstation. To permit evaluating the proposed retiming algorithm, modified Microelectronic Center of North Carolina (MCNC) benchmark circuits [25], [26] have been analyzed with this algorithm and an implementation of the Leiserson-Saxe retiming algorithm [1]. The resulting minimum clock period for each of the retimed benchmark circuits is reported.

To evaluate the proposed retiming algorithm, 1989 and 1991 MCNC LGSynth benchmark circuits [25] and [26] have been modified to include the effects of variable register, clock, and interconnect delays. To incorporate these delay components into the benchmark circuits, REC's are artificially generated using a random number generator. However, to better simulate the effects of the actual clock distribution, interconnect, and register delays, the uniformly distributed numbers generated by the C library function **random** () are converted to a normal Gaussian distribution [27]. For clock delays, a uniform distribution is applied, since the registers are typically distributed over the entire integrated circuit. Physically distant registers may have very different clock delays, since the interconnect impedance between the clock source and these registers and the capacitive loading of the registers may vary over a wide range. This suggests a wide spectrum of clock delay values, and, therefore, a uniform distribution is applied. For interconnect delays, a uniform distribution is also used, since the distance between the registers and the logic elements is assumed to vary uniformly. A Gaussian distribution is used for the register delays, since similar instances of the same register cell or register cells of similar delay are most often used; therefore, the delays are approximated as being normally distributed. For those instances in which a negative value for the clock, register, or interconnect delays is obtained from the Gaussian distribution, the approach applied in these experiments is to discard the negative values and redo the sample. The "truncation toward zero" approach (i.e., mapping of negative values to zero) is not applied since this would bias the probability of obtaining zero values (specifically, the probability of obtaining a zero sample would be the integral of the Gaussian distribution from $-\infty$ to zero).

As described in Section IV, a process-dependent parameter k is used to prevent marginal race conditions. Paths with a delay less than or equal to this parameter are not permitted. In the benchmark circuits, $k = 0$ is used, therefore, paths with zero or negative delay are not permitted. In these circuits, the average register delay $(T_{C\rightarrow Q} + T_{\text{Set-up}})$ of each circuit is added to each local data path to compensate for the effects of the variable register delays, i.e., the register delay of each local data path is assumed to be constant and equal to the average register delay of the retimed circuit with variable REC values.

The application of *RETSAM* to the example MCNC benchmark circuits are described in Table III. The initial five

TABLE III
RESULTS OF THE APPLICATION OF THE RETIMING ALGORITHM TO MCNC BENCHMARK CIRCUITS

Example	Graph properties				T_{CP} After Retiming (tu)	T_{CP} T_{SKEW}=0 T_{REG}=const (tu)	CPU Time	
	Edges	Vertices	Latency	Initial T_{CP} (tu)			LP Based (sec)	*RETSAM* (sec)
LGSynth89 - multi-level (netblif)								
C17	26	19	6	92	29	25	2.44	3499
b1	34	19	5	80	33	28	4.75	129
cm138a	62	29	4	110	43	38	5.93	11555
cm42a	65	34	3	101	46	38	6.71	461
cm82a	59	37	4	139	47	40	5.76	4260
majority	26	17	5	103	33	35	4.00	1045
LGSynth91 - multi level (blif)								
C17	19	12	5	63	32	26	2.85	160
b1	16	10	2	50	33	24	1.30	3.03
cm42a	49	18	4	78	35	32	1.38	351
cm82a	22	12	3	49	28	26	5.61	10.4
cm85a	70	36	3	102	51	40	5.24	11573
cm150a	69	38	2	90	56	48	4.35	94.9
cm151a	38	22	3	93	41	36	2.22	2850
decod	89	24	4	69	43	37	4.54	61996
parity	47	32	2	77	49	36	4.42	35.5
tcon	65	34	2	40	34	24	3.56	34.0
Figure 2 with the added REC values								
fig2	11	8	4	43*	24	20	1.171	2.172

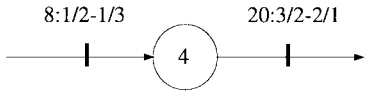
* denotes a graph that contains race conditions before retiming.

columns describe the properties of the modified benchmark circuits. These properties are:

1) the name of the benchmark example as it appears in the MCNC archive;
2) the number of edges;
3) vertices in the graph of each circuit;
4) the latency of the circuit;
5) the original clock period.

The minimum clock period of the retimed circuit using algorithm *RETSAM* is shown in the sixth column. In the seventh column, the minimum T_{CP} biased with T_{Reg} using algorithm *FEAS* [1] is presented. The parameter T_{Reg} is the average register and interconnect delay in the circuit and is included to provide a fairer comparison.

The sequential circuit represented by the retimed graph contains R registers with various REC values. If the register delays in the circuit are different, as shown in Fig. 11 ($T_{C\rightarrow Q} + T_{\text{Set-up}} = 2 + 1$ tu vs. $2 + 3$ tu), $T_{\text{Set-up}}$ for the initial register and $T_{C\rightarrow Q}$ for the final register must be considered when calculating the path delay. In the unusual case where all register delays are equal, T_{Reg} can be used as a global parameter, as is assumed in [3]. The greater the variance between the register delays and between the clock delays (the more significant the affect of negative clock skew), the greater the minimum clock period becomes, as exemplified by the minimum clock period of certain benchmark circuits listed in the sixth column. This increased delay is due to the *imbalance* among the path delays, thereby increasing the worst case path delay, requiring a larger minimum clock period. Since the circuits listed in Table III are relatively balanced, negative clock skew does not significantly impact most of the circuits listed in Table III. Thus, only one circuit (the majority circuit) exhibits a minimum clock period (listed in column six) which is less than the clock period shown in column seven.



$$T_{PD} = T_{Reg} + T_{Int} + T_{Logic} + T_{Skew}$$
$$T_{PD} = (2+3) + (3+2) + \quad 4 \quad + (8-20)$$
$$T_{PD} = 2\ tu$$

Fig. 11. A path containing two registers and a vertex between the two registers. The path delay T_{PD} contains components related to the register delays. If all registers in the circuit are similar, the register delay components would be equal.

To provide a comparison between the central processing unit (CPU) efficiency of the retiming process on a SUN4 workstation with and without monotonic delays, the CPU times are included in columns 8 and 9 of Table III. The CPU times required to retime a circuit with monotonic path delays using a linear programming based algorithm similar to *FEAS* proposed in [1] is listed in column 8. The CPU times using *RETSAM* are listed in column 9. Note the dramatic improvement in CPU efficiency when a linear programming based algorithm is used. This emphasizes the importance of applying path delay monotonicity constraints when retiming a high complexity circuit.

As shown in Table III and noted earlier, the minimum clock period of the majority circuit from LGSynth89 retimed with *RETSAM* is less than from existing retiming algorithms. This occurs since localized negative clock skew [5] and [18] subtracts delay from the critical path such that the worst case path delay is smaller, thereby causing the minimum clock period to be less. Also, note that no race conditions exist in those circuits retimed by *RETSAM*, a conclusion that cannot be drawn with other retiming algorithms.

IX. Conclusions

A retiming algorithm is presented which considers variable clock distribution, register, and interconnect delays. To permit the consideration of these delay components, REC's are attached to each edge of the graph representing the circuit, and the original path delays are redefined to be from edge-to-edge rather than vertex-to-vertex. A set of inequalities are created based on these edge-to-edge path delays, permitting a retimed version of the circuit to operate at the minimum clock period. A general algorithm, *RETSAM*, is presented which supports arbitrary REC values, including excessive negative clock skew. An iterative method using ranges of vertex lags rather than constant vertex lags is used within this retiming algorithm to solve for the edge weights.

A set of monotonicity conditions may be imposed on the REC values to improve the computational efficiency by permitting the use of standard linear programming methods. These monotonicity conditions place constraints on the magnitude of the negative clock skew of each local data path, thereby no longer permitting the clock delays to be of arbitrary value. The feasibility of applying these conditions to practical circuits is discussed. It is shown that retiming cannot be efficiently and accurately performed on a circuit with an improperly designed clock distribution network. Thus, the quality of the design of the clock distribution network can significantly affect the automated design of high-performance synchronous circuits when utilizing retiming as a synthesis methodology.

The limitations and advantages of the proposed retiming algorithm are compared with existing retiming strategies using a set of modified MCNC benchmark circuits. The results of applying *RETSAM* to the benchmark circuits show that a more accurate retiming can be performed than with existing retiming algorithms which do not consider variable clock distribution, register, and interconnect delays. Additionally, the clock period can be further minimized due to localized negative clock skew. Finally, clock skew induced race conditions are detected and eliminated.

Summarizing, a new retiming algorithm which considers the effects of variable clock distribution, register, and interconnect delays has been presented. This algorithm represents a significant extension of existing retiming algorithms, permitting the use of retiming for the automated synthesis of higher speed and more reliable pipelined digital systems.

Appendix A
Estimating the Clock Delays

As described in Section I, the retiming process requires an initial estimate of the register, clock distribution, and interconnect delays which can be replaced with more accurate values as the exploratory retiming process becomes better specified. Therefore, an important aspect of this research effort is that the retiming process and the clock distribution design are closely related, i.e., a careful design of the clock distribution network will improve the results of the retiming process. The retiming process can also be used to improve the design of the clock distribution network.

Although optimal retiming and clock distribution design may be performed separately, simultaneous clock distribution design and retiming may be highly CPU intensive. Therefore, estimating the delay characteristics of the clock distribution network and integrating these delays into the retiming process may be more suitable for designing practical circuits. This issue is a topic of considerable focus within the academic and industrial research communities.

A procedure is described to demonstrate how clock delays can be estimated for a practical circuit. As previously mentioned, it is assumed that an integrated circuit can be partitioned into regions of similar clock delay. Each clock delay region is composed of both a deterministic and a probabilistic delay component, as shown below

$$T_{CD}(region = n) = T_{CD_{\text{const}}}(region = n) + \mathcal{T} \qquad \text{(A.1)}$$

where $\mathcal{T}$ is a uniformly-distributed random variable, and $T_{CD_{\text{const}}}(region = n)$ is the deterministic clock delay attached to region n.

The nondeterministic statistically based component of the clock path delay, represented by the delay component $\mathcal{T}$, is due to variations in process parameters within the integrated circuit. Since transistor parameters, such as channel mobility, threshold voltage, and oxide thickness, may vary across the die, some variations of the clock delay are expected. This variation, however, tends to be small for registers belonging to the same local data path (and, therefore, physically close to each other). Methods have been proposed in the literature for reducing the process dependence of the clock skew to less than 10% of the total path delay [28].

The deterministic component of the clock delay for a region n $[T_{CD_{\text{const}}}(region = n)]$ can be calculated based on geometric distance information derived from the circuit layout. The distance information from a source node to any sink node in an integrated circuit (IC) layout can be obtained either from routing information or using exploratory Steiner

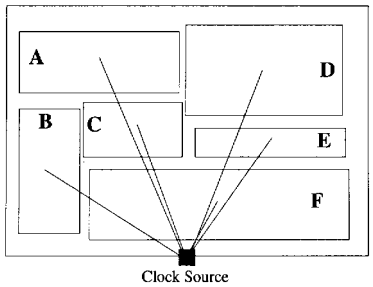


Fig. 12. A block diagram of an example integrated circuit layout. The chip area is assumed to be partitioned into regions of similar clock delay.

tree approaches [29], [30]. Calculation of the interconnect resistances and capacitances directly from the IC layout or Steiner tree can be achieved with standard circuit extraction techniques [31]. With this information, well-known techniques can be employed to estimate the individual clock delays (e.g., [13], [32]–[35]).

As observed from (A.1), each region n is attached a deterministic clock delay $T_{CD_{\text{const}}}$ characterizing that region. A floorplan of a simple integrated circuit consisting of six clock delay regions is depicted in Fig. 12. As an example, assume that the total interconnect capacitance of the clock line driving region A is $C_{\text{int}} = 300$ fF, the total interconnect resistance of the clock line is $R_{\text{int}} = 200\ \Omega$, the on-resistance of the clock source is $R_{tr} = 300\ \Omega$, and this clock line drives 100 registers (four NMOS and four PMOS devices per register). The geometric size of the NMOS and PMOS devices are 10 μm $\times$ 1 μm and 20 μm $\times$ 1 μm, respectively. Therefore, the total load capacitance C_L of the module A can be approximated by (ignoring bulk capacitance)

$$C_L = 100 \times 4(W_P L_P + W_N L_N) \cdot \frac{\epsilon_{\text{ox}}}{t_{\text{ox}}} = 21\ \text{pF} \qquad \text{(A.2)}$$

where W_P and W_N are the gate widths of the PMOS and NMOS load transistors, L_P and L_N are the gate lengths of the PMOS and NMOS load transistors, ϵ_{ox} is the permitivity of the gate oxide, and t_{ox} is the gate oxide thickness (a value of 200 Å is assumed). Therefore, the deterministic component of the clock delay (defined at the 50% point) from the clock source to module A can be estimated as [33], [34]

$$\begin{aligned} &T_{CD_{\text{const}}}(region = A) \\ &\quad = 0.4 R_{\text{int}} C_{\text{int}} + 0.7(R_{tr} C_{\text{int}} + R_{tr} C_L + R_{\text{int}} C_L) \\ &\quad = 0.4 \times 60\ \text{ps} + 0.7(90 + 6300 + 4200)\ \text{ps} \approx 7.4\ \text{ns}. \end{aligned} \qquad \text{(A.3)}$$

This simple example presents an initial approach for estimating the delays within a clock distribution network. Research is currently under way for developing more accurate models for estimating these REC values. More sophisticated approaches for estimating IC area and performance characteristics can be found in [36]–[38].

Appendix B
Derivation of the Monotonicity Constraints

In Section VII, the importance of monotonically increasing path delays is described. Computationally efficient retiming cannot be guaranteed on a synchronous circuit with arbitrary clock delays yielding arbitrary data path delays. It may be preferable to design the clock distribution network to improve the computational efficiency of the retiming algorithm by ensuring that all path delays are monotonically increasing. In Section VII-B, inequalities are presented that must be satisfied by each individual data path to ensure monotonically increasing path delays. The derivation of these inequalities, (29) and (30), are provided in this appendix.

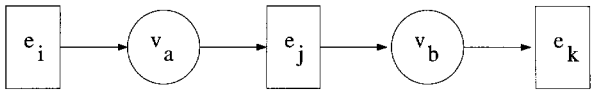


Fig. 13. A path p with three registers and two vertices.

In Fig. 13, a path with three registers and two vertices with delays $d(v_a)$ and $d(v_b)$ is depicted. For this path consisting of three registers, i, j, and k, necessary conditions for monotonically increasing path delays (smaller delays for subpaths of larger paths) are

$$T_{PD}(i,k) \geq T_{PD}(i,j) \qquad \text{(B.1)}$$

$$T_{PD}(i,k) \geq T_{PD}(j,k). \qquad \text{(B.2)}$$

To provide intuition into how these inequalities are created, consider the graph of Fig. 9. In this figure, (B.1) and (B.2) can be used to ensure that the subpaths p_1 and p_2 each have a delay less than the longer path p. More generally, (B.1) and (B.2) ensure that paths $e_i \rightsquigarrow e_j$ and $e_j \rightsquigarrow e_k$ each have a delay less than the longer path $e_i \rightsquigarrow e_k$.

Using (4), the following inequalities can be derived from (B.1) and (B.2)

$$\begin{aligned} &T_{C\to Q}(e_i) + T_{\text{Int2}}(e_i) + d(v_a) + T_{\text{Int1}}(e_j) + T_{\text{Int2}}(e_j) \\ &\qquad + d(v_b) + T_{\text{Int1}}(e_k) + T_{\text{Set-up}}(e_k) \\ &\qquad + T_{\text{Skew}}(e_i, e_k) \geq T_{C\to Q}(e_i) + T_{\text{Int2}}(e_i) \\ &\qquad + d(v_a) + T_{\text{Int1}}(e_j) + T_{\text{Set-up}}(e_j) + T_{\text{Skew}}(e_i, e_j) \end{aligned} \qquad \text{(B.3)}$$

$$\begin{aligned} &T_{C\to Q}(e_i) + T_{\text{Int2}}(e_i) + d(v_a) + T_{\text{Int1}}(e_j) + T_{\text{Int2}}(e_j) \\ &\qquad + d(v_b) + T_{\text{Int1}}(e_k) + T_{\text{Set-up}}(e_k) \\ &\qquad + T_{\text{Skew}}(e_i, e_k) \geq T_{C\to Q}(e_j) + T_{\text{Int2}}(e_j) \\ &\qquad + d(v_b) + T_{\text{Int1}}(e_k) + T_{\text{Set-up}}(e_k) + T_{\text{Skew}}(e_j, e_k). \end{aligned} \qquad \text{(B.4)}$$

Defining $T_{\text{REG}}(a)$ and $T_{\text{REG}}(b)$ as the total setup and clock-to-Q delays of edges a and b, respectively

$$\begin{aligned} T_{\text{REG}}(a) &= T_{\text{Set-up}}(a) + T_{C\to Q}(a) \\ T_{\text{REG}}(b) &= T_{\text{Set-up}}(b) + T_{C\to Q}(b) \end{aligned} \qquad \text{(B.5)}$$

the following inequalities are obtained from (B.3) and (B.4)

$$T_{PD}(j,k) \geq T_{\text{REG}}(j) \qquad \text{(B.6)}$$

$$T_{PD}(i,j) \geq T_{\text{REG}}(j). \qquad \text{(B.7)}$$

These two conditions, (B.6) and (B.7), can be transformed into a simpler form by combining them. Assume a path with two edges a and b, respectively, and a vertex between these

edges. The following condition is required to ensure that the path delays increase monotonically with increasing path length

$$T_{PD}(a,b) \geq \max\{T_{\mathrm{REG}}(a), T_{\mathrm{REG}}(b)\}. \tag{B.8}$$

Equation (B.8) sets a lower limit on the delay of a local data path to ensure that only one inequality is required, making the system of inequalities linear, thereby permitting the Bellman–Ford method to be used. There is also an upper limit that can be defined as

$$T_{PD}(a,b) \leq C \tag{B.9}$$

where C is the maximum permitted clock period of the circuit. This upper limit is used to guarantee that each local data path has a delay smaller than the maximum permitted clock period of the synchronous circuit. Since monotonicity is guaranteed by (B.8), a longer path will have a greater delay. Therefore, the upper limit of (B.9) must be imposed on each local data path, since if (B.9) is not satisfied for each local data path, the excessively long local data path delay will place a new lower limit on the clock period of the circuit. If this upper limit is greater than required, the long path must be removed.

Since $T_{PD}(a,b)$ depends on the clock delays driving the initial and final registers of the local data path between edges e_a and e_b, (B.8) is a constraint which is imposed on the clock distribution network. Equation (B.8) can be expanded into (B.10) and (B.11). Equation (B.11) describes a specific constraint that must be placed on the individual clock delays

$$\begin{aligned} &T_{CD}(a) - T_{CD}(b) + T_{C\to Q}(a) + T_{\mathrm{Int2}}(a) + d(v_a) \\ &\qquad + T_{\text{Set-up}}(b) + T_{\mathrm{Int1}}(b) \\ &\quad \geq \max\{T_{\mathrm{REG}}(a), T_{\mathrm{REG}}(b)\} \end{aligned} \tag{B.10}$$

$$T_{CD}(a) - T_{CD}(b) \geq -\tau(a,b). \tag{B.11}$$

In (B.11), $\tau(a,b)$ denotes a constant depending only on the REC parameters of edges a and b. $\tau(a,b)$ can be calculated from (B.12) and is the "**negative clock skew tolerance of the local data path from edge a to edge b**"

$$\begin{aligned} \tau(a,b) = &T_{C\to Q}(a) + T_{\mathrm{Int2}}(a) + d(v_a) + T_{\text{Set-up}}(b) \\ &+ T_{\mathrm{Int1}}(b) - \max\{T_{\mathrm{REG}}(a), T_{\mathrm{REG}}(b)\}. \end{aligned} \tag{B.12}$$

Note that in the more general case where the process dependent parameter k is nonzero, k must also be included in (B.12), i.e., $\tau(a,b) + k$ is used rather than $\tau(a,b)$ as the negative clock skew tolerance. Also note that since k is always greater than zero, the negative clock skew tolerance of the local data paths decreases with increasing values of k, thereby further constraining the design of the clock distribution network.

Another interpretation of (B.11) and (B.12) is that "the clock skew of any local data path cannot be more negative than the clock skew tolerance of that local data path." An observation of (B.8) is that strictly positive local data path delays are required, i.e., "computationally inexpensive retiming" cannot be performed with the existence of race conditions. This characteristic is a profound result in that it states that computationally efficient and accurate retiming cannot be performed on a circuit with an inefficiently designed clock distribution network. In other words, the process of retiming and clock distribution network design are closely related. The clock distribution network must be designed to satisfy (B.11). For computational efficiency, retiming is best performed on a synchronous digital system with a properly designed clock distribution network in which the negative clock skew of a local data path does not exceed the maximum tolerance of that path, as determined by (B.12).

Assuming (B.11) is satisfied, it may be possible to design a clock distribution network which permits computationally efficient retiming. Equations (B.11) and (B.9) can be rewritten as (B.13) and (B.14), respectively

$$T_{CD}(b) - T_{CD}(a) \leq \tau(a,b) \tag{B.13}$$

$$T_{CD}(a) - T_{CD}(b) \leq T(a,b) \tag{B.14}$$

where

$$\begin{aligned} T(a,b) = &C - T_{C\to Q}(a) - T_{\mathrm{Int2}}(a) - d(v_a) \\ &- T_{\text{Set-up}}(b) - T_{\mathrm{Int1}}(b). \end{aligned} \tag{B.15}$$

$T(a,b)$ is the "**positive clock skew tolerance of the local data path from edge a to edge b**."

ACKNOWLEDGMENT

The authors of this paper would like to thank Dr. J. Saxe for his assistance on certain questions regarding the theory of retiming. The first two authors of this paper would like to specifically acknowledge the insightful and important contributions of the third author and their mentor and friend, Dr. James H. Mulligan, Jr., who passed away on January 12, 1996.

REFERENCES

[1] C. E. Leiserson and J. B. Saxe, "Retiming synchronous circuitry," *Algorithmica*, vol. 6, pp. 5–35, Jan. 1991.
[2] A. T. Ishii, C. E. Leiserson, and M. C. Papaefthymiou, "Optimizing two-phase, level-clocked circuitry," in *Proc. Conf. Advanced Research VLSI and Parallel Systems*, Mar. 1992, pp. 245–264.
[3] G. DeMicheli, "Synchronous logic synthesis: algorithms for cycle-time minimization," *IEEE Trans. Computer-Aided Design*, vol. 10, no. 1, pp. 63–73, Jan. 1991.
[4] J. P. Fishburn, "Clock skew optimization," *IEEE Trans. Comput.*, vol. 39, no. 7, pp. 945–951, July 1990.
[5] E. G. Friedman, "The application of localized clock distribution design to improving the performance of retimed sequential circuits," in *Proc. IEEE Asia-Pacific Conf. Circuits and Systems*, Dec. 1992, pp. 12–17.
[6] T. Soyata, E. G. Friedman, and J. H. Mulligan, Jr., "Integration of clock skew and register delays into a retiming algorithm," in *Proc. IEEE Int. Symp. Circuits and Systems*, May 1993, pp. 1483–1486.
[7] B. Lockyear and C. Eberling, "The practical application of retiming to the design of high-performance systems," in *Proc. IEEE Int. Conf. Computer-Aided Design*, Nov. 1993, pp. 288–295.
[8] T. Soyata and E. G. Friedman, "Synchronous performance and reliability improvement in pipelined ASIC's," in *Proc. IEEE ASIC Conf.*, Sept. 1994, pp. 383–390.
[9] T. Soyata and E. G. Friedman, "Retiming with nonzero clock skew, variable register, and interconnect delay," in *Proc. IEEE Int. Conf. Computer-Aided Design*, Nov. 1994, pp. 234–241.
[10] S. Simon, E. Bernard, M. Sauer, and J. A. Nossek, "A New retiming algorithm for circuit design," in *Proc. IEEE Int. Symp. Circuits and Systems*, May/June 1994, pp. 4.35–4.38.
[11] L. Chao and E. H. Sha, "Retiming and clock skew for synchronous systems," in *Proc. IEEE Int. Symp. Circuits and Systems*, May/June 1994, pp. 1.283–1.286.

[12] J. L. Neves and E. G. Friedman, "Topological design of clock distribution networks based on nonzero clock skew specifications," in *Proc. IEEE Midwest Symp. Circuits and Systems*, Aug. 1993, pp. 468–471.
[13] J. L. Neves and E. G. Friedman, "Circuit synthesis of clock distribution networks based on nonzero clock skew," in *Proc. IEEE Int. Symp. Circuits and Systems*, May/June 1994, pp. 4.175–4.179.
[14] J. L. Neves and E. G. Friedman, "Synthesizing distributed buffer clock trees for high-performance ASIC's," in *Proc. IEEE ASIC Conf.*, Sept. 1994, pp. 126–129.
[15] J. L. Neves and E. G. Friedman, "Design methodology for synthesizing clock distribution networks exploiting nonzero localized clock skew," *IEEE Trans. VLSI Syst.*, vol. 4, no. 2, June 1996.
[16] T. Soyata, E. G. Friedman, and J. H. Mulligan, Jr., "Monotonicity constraints on path delays for efficient retiming with localized clock skew and variable register delay," in *Proc. IEEE Int. Symp. Circuits and Systems*, May 1995, pp. 1748–1751.
[17] E. G. Friedman and J. H. Mulligan, Jr., "Clock frequency and latency in synchronous digital systems," *IEEE Trans. Signal Processing*, vol. 39, no. 4, pp. 930–934, Apr. 1991.
[18] E. G. Friedman, "Clock distribution design in VLSI circuits—an overview," in *Proc. IEEE Int. Symp. Circuits and Systems*, May 1993, pp. 1475–1478.
[19] E. G. Friedman, *Clock Distribution Networks in VLSI Circuits and Systems*. Piscataway, NJ: IEEE Press, 1995.
[20] K. A. Sakallah, T. N. Mudge, T. M. Burks, and E. S. Davidson, "Synchronization of pipelines," *IEEE Trans. Computer-Aided Design*, vol. 12, no. 8, pp. 1132–1146, Aug. 1993.
[21] E. L. Lawler, *Combinatorial Optimization: Networks and Matroids*. New York: Holt, Rinehart and Winston, 1976.
[22] D. B. Johnson, "Efficient algorithms for shortest paths in sparse networks," *J. Assoc. Comput. Mach.*, vol. 24, no. 1, pp. 1–13, 1977.
[23] T. H. Cormen, C. E. Leiserson, and R. L. Rivest, *Introduction to Algorithms*. New York: McGraw-Hill, 1990.
[24] S. Malik, E. M. Sentovich, R. K. Brayton, and A. Sangiovanni-Vincentelli, "Retiming and resynthesis: optimizing sequential networks with combinatorial techniques," *IEEE Trans. Computer-Aided Design*, vol. 10, no. 1, pp. 74–84, Jan. 1991.
[25] R. Lisanke, "Logic synthesis and optimization benchmarks user guide: version 2.0," Microelectronics Center North Carolina, Tech. Rep., Dec. 1988.
[26] S. Yang, "Logic synthesis and optimization benchmarks user guide: version 3.0," Microelectronics Center North Carolina, Tech. Rep., Jan. 1991.
[27] W. H. Press, B. P. Flannery, S. A. Teukolsky, and W. T. Vetterling, *Numerical Recipes in C*. Cambridge: Cambridge Univ. Press, 1990.
[28] M. Shoji, "Elimination of process-dependent clock skew in CMOS VLSI," *IEEE J. Solid-State Circuits*, vol. SC-21, no. 5, pp. 875–880, Oct. 1986.
[29] J. Cong, A. Kahng, G. Robins, M. Sarrafzadeh, and C. K. Wong, "Performance-driven global routing for cell based IC's," in *Proc. IEEE Int. Conf. Computer Design*, Oct. 1991, pp. 170–173.
[30] T. Chao, Y. Hsu, J. Ho, K. D. Boese, and A. B. Kahng, "Zero clock skew routing with minimum wirelength," *IEEE Trans. Circuits and Syst. II*, vol. 39, no. 11, pp. 799–814, Nov. 1992.
[31] W. S. Scott and J. K. Ousterhout, "Magic's circuit extractor," *IEEE Design Test Comput.*, vol. 3, no. 1, pp. 24–34, Feb. 1986.
[32] J. Rubinstein, P. Penfield, and M. A. Horowitz, "Signal delay in RC tree networks," *IEEE Trans. Computer-Aided Design*, vol. CAD-2, no. 3, pp. 202–211, July 1983.
[33] T. Sakurai, "Approximation of wiring delay in MOSFET VLSI," *IEEE J. Solid-State Circuits*, vol. SC-18, no. 4, pp. 418–426, Aug. 1983.
[34] H. B. Bakoglu, *Circuits Interconnections, and Packaging for VLSI*. Reading, MA: Addison Wesley, 1990.
[35] T. Sakurai, "Closed-form expressions for interconnect delay, coupling, and crosstalk in VLSI's," *IEEE Trans. Electron Devices*, vol. 40, no. 1, pp. 118–124, Jan. 1993.
[36] C. Ramachandran and F. J. Kurdahi, "Combined topological and functionality based delay estimation using layout-driven approach for high level applications," in *Proc. ACM/IEEE European Design Automation Conf.*, Sept. 1992, pp. 72–78.
[37] C. Ramachandran, F. J. Kurdahi, D. D. Gajski, A. C. H. Wu, and V. Chaiyakul, "Accurate layout area and delay modeling for system level design," in *Proc. IEEE Int. Conf. on Computer-Aided Design*, Nov. 1992, pp. 355–361.
[38] S. D. Rao and F. J. Kurdahi, "Hierarchical design space exploration for a class of digital systems," *IEEE Trans. VLSI Syst.*, vol. 1, no. 3, pp. 282–295, Sept. 1993.

Tolga Soyata (S'96–M'96) received the B.S.E.E. degree from Istanbul Technical University, Istanbul, Turkey, in 1988, the M.S.E.C.E. degree from the Johns Hopkins University, Baltimore, MD, in 1992, and the M.S.E.E. degree from the University of Rochester, Rochester, NY, in 1993. He is presently working toward the Ph.D. degree in electrical engineering at the University of Rochester.

His research interests include high performance VLSI/IC design using pipelining, retiming, and clock skew optimization. He has published four conference papers in the area of high performance VLSI circuit optimization.

Eby G. Friedman (S'85–M'89–SM'90) was born in Jersey City, NJ, in 1957. He received the B.S. degree from Lafayette College, Easton, PA, in 1979 and the M.S. and Ph.D. degrees in electrical engineering from the University of California, Irvine, in 1981 and 1989, respectively.

He was with Philips Gloeilampen Fabrieken, Eindhove, The Netherlands, in 1978 where he worked on the design of bipolar differential amplifiers. From 1979 to 1991, he was with Hughes Aircraft Company, rising to the position of Manager of the Signal Processing Design and Test Department, responsible for the design and test of high performance digital and analog IC's. He has been with the Department of Electrical Engineering at the University of Rochester, Rochester, NY, since 1991, where he is an Associate Professor and Director of the High Performance VLSI/IC Design and Analysis Laboratory. His current research and teaching interests are in high performance microelectronic design and analysis with application to high speed portable processors and low power wireless communications.

He has authored two book chapters and many papers in the fields of high-speed and low-power CMOS design techniques, pipelining and retiming, and the theory and application of synchronous clock distribution networks, and has edited one book *Clock Distribution Networks in VLSI Circuits and Systems* (Piscataway, NJ: IEEE Press, 1995).

Dr. Friedman is a member of the editorial board of *Analog Integrated Circuits and Signal Processing*, Chair of the VLSI Systems and Applications CAS Technical Committee, Chair of the VLSI track for ISCAS '96 and '97, and a member of the technical program committee of a number of conferences. He was a member of the editorial board of the IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS II: ANALOG AND DIGITAL SIGNAL PROCESSING, Chair of the Electron Devices Chapter of the IEEE Rochester Section, and a recipient of the Howard Hughes Masters and Doctoral Fellowships, an NSF Research Initiation Award, an Outstanding IEEE Chapter Chairman Award, and a University of Rochester College of Engineering Teaching Excellence Award.

James H. Mulligan, Jr. received the B.E.E. and E.E. degrees from the Cooper Union, New York, in 1943 and 1947, respectively, the M.S. degree in electrical engineering from Stevens Institute of Technology, Hoboken, NJ, in 1945, and the Ph.D. degree in electrical engineering from Columbia University, New York, in 1948.

His career includes engineering responsibilities in industrial, government, and academic organizations. He was employed by Bell Laboratories, the Naval Research Laboratory, and the Allen B. DuMont Laboratories. From 1949 to 1968 he was a member of the faculty of the Department of Electrical Engineering, New York University, New York, serving as Chairman of the Department from 1952 to 1968. From 1968 to 1974 he was Secretary and Executive Officer of the National Academy of Engineering. He served as Dean of the School of Engineering at the University of California, Irvine, from 1974 to 1977, following which he returned to full-time teaching and research as Professor of Electrical Engineering at that institution. He passed away on January 12, 1996.