Retiming With Non-Zero Clock Skew, Variable Register, and Interconnect Delay

Tolga Soyata and Eby G. Friedman

Department of Electrical Engineering
University of Rochester, Rochester, NY 14627

Abstract

A retiming algorithm is presented which includes the effects of variable register, clock distribution, and interconnect delay. These delay components are incorporated into retiming by assigning Register Electrical Characteristics (RECs) to each edge in the graph representation of the synchronous circuit. A matrix (called the Sequential Adjacency Matrix or 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 standard retiming algorithms. The approach used in the proposed algorithm is to initialize these ranges with unbounded values and continuously tighten these ranges using localized timing constraints until an optimal solution is obtained. The algorithm is demonstrated on modified MCNC benchmark circuits and both increased clock frequencies and elimination of all race conditions are observed.

1 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. In retiming algorithms proposed to date, variable register, clock distribution, and interconnect delays are essentially ignored. Without including these delay components, standard retiming algorithms are not sufficiently accurate for building practical high speed circuits. For this reason, clock distribution, variable register, and interconnect delay must be integrated into the retiming process in order to ensure that retiming becomes a practical and useful design methodology.

Both register and interconnect delay 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.

This research is based upon work supported by the National Science Foundation under Grant No. MIP-9208165.

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 set-up (t_s) and hold (t_p) times are non-zero 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 may have a different register delay.

Integrating clock skew into the retiming process was first proposed in [4, 5]. The authors of this paper originally introduced the strategy of integrating clock skew and variable register delays into retiming by attaching electrical information to the edges of the graph representing the synchronous circuit [6]. These delay parameters are defined as Register Electrical Characteristics (RECs) 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 non-zero clock skew, variable register, and interconnect delay. In general, there has been a growing interest in making retiming into a more practical and useful design methodology, evidenced by [1–10].

In this paper, a retiming algorithm is presented which incorporates variable register and interconnect delay and non-zero localized clock skew. Either rising edge or falling edge triggered D flip flops and a single phase clock is 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

©1994 ACM 0-89791-690-5/94/0011/0234 $3.50

both maximum clock periods and race conditions to be detected on a path-by-path basis. This approach, therefore, initially requires approximate (or estimated) values of register, clock distribution, and interconnect delays which can be replaced with more accurate values as the exploratory retiming process becomes better specified [6, 8].

The paper is organized as follows. Background and definitions of important terms used throughout the paper are provided in Section 2. In Section 3, models of non-zero clock skew, variable register, and interconnect delay are presented. In Section 4, the Sequential Adjacency Matrix (SAM) is introduced. Timing constraints, derived from the SAM, are described in Section 5. The proposed retiming algorithm RETSAM is presented in Section 6. Results of applying the proposed algorithm to MCNC benchmark circuits are presented in Section 7 and finally some conclusions are drawn in Section 8.

2 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 [11]. They show that only 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 path sequentially adjacent.

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

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

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.

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, 12], 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, 11, 12]. Those paths with negative delay are called **short paths** [13]. 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 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 connection 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 $e_k.start$ and $e_k.end$.

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 in this paper. Using this definition, retiming can be defined as assigning a lag to each vertex using the following formula,

$$w_r(e) = w(e) + r(u) - r(v), \tag{2}$$

where e is an edge connecting vertices u and v and $w(e)$ and $w_r(e)$ are the weight of edge e before and after retiming, respectively. 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

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

3 Register Electrical Characteristics (RECs)

In order to consider the effects of clock distribution, variable register, and interconnect delay, a number set, the **Register Electrical Characteristic** (REC), is assigned to each edge of the graph in the following form: $T_{CD} : T_{Set-up}/T_{C\rightarrow Q} - T_{Int1}/T_{Int2}$. T_{CD} is the clock delay from the global clock source to each register, T_{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, 8] rather than vertex-to-vertex, as in standard retiming algorithms [1]. A modified version of the graph introduced in [1] is shown in Figure 1, in which an REC is assigned to each edge. By assigning a clock delay to each edge, 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 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 local data path delay $T_{PD}(i,j)$ from edges e_i to e_j is

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

where $T_{Logic}(i,j)$ 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.

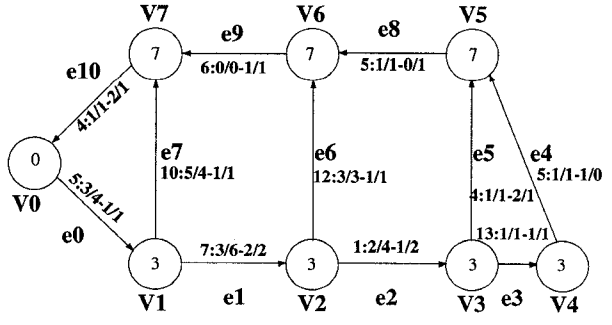

Figure 1: Graph of the digital correlator [1] with added REC values

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 RECs, 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 $T_{PD_{Internal}}(e_k)$, given by

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

4 Sequential Adjacency Matrix (SAM)

The Sequential Adjacency Matrix (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

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

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'.

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

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

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

Note that the S' matrix contains information for only those paths that can potentially cause the circuit to function improperly. Therefore, the 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 these negative entries flag catastrophic race conditions. 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 Figure 1 is shown in Table 1. 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 23 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 a catastrophic race condition.

5 Timing Constraints

In this paper, a branch and bound algorithm is used 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 constraints. These different types of constraints are explained in greater detail in the following subsections.

5.1 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 [14] 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. The negative edge weight constraint can be written as

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

Table 1: SAM for the graph of Figure 1. Light shaded entries represent short paths, whereas dark shaded entries represent long paths for $c = 23$ tu. Unshaded entries denote permissible paths.

SAM		to e0	e1	e2	e3	e4	e5	e6	e7	e8	e9	e10
from	e0	24	11	22	15	28	25	12	9	32	24	21
	e1	38	42	20	13	26	23	10	40	30	22	35
	e2	39	43	34	-1	12	9	34	41	16	23	36
	e3	45	49	60	53	15	63	50	47	22	26	42
	e4	42	46	47	40	53	50	37	34	9	16	29
	e5	42	46	47	40	53	50	37	34	9	16	29
	e6	54	58	49	42	55	57	39	36	59	18	31
	e7	24	28	39	32	45	42	29	26	49	41	21
	e8	25	29	40	33	46	43	30	27	50	9	22
	e9	16	20	31	24	37	34	21	18	41	33	13
	e10	5	9	20	13	26	23	10	7	30	22	19

5.2 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 1, long paths for $c = 23$ 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). These three conditions can be written in terms of path and edge weights as follows:

$$w(e_i) = 0, \tag{11}$$

$$w(e_j) = 0, \tag{12}$$

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

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), (12), and (13) that either the initial or final edge does not have any register located on it or there is at least one register along every path between these two edges.

5.3 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 the final registers of this path must be made nonsequentially adjacent. Equations (11), (12), and (13) are used to eliminate any catastrophic short paths (or race conditions).

5.4 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. Internal long path constraints can be formulated using (5) as

$$w(e_i) \leq 1, \ \forall\, i : T_{PD_{internal}}(e_i) > c. \qquad (14)$$

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

5.5 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.

$$r(e.start) - r(e.end) \leq w(e), \ \forall e \in E, \qquad (15)$$

$$r(e_i.start) - r(e_i.end) = w(e_i), \qquad (16)$$

$$r(e_j.start) - r(e_j.end) = w(e_j), \qquad (17)$$

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

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

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.

6 Retiming Algorithm

In this section two algorithms are introduced: 1) Algorithm *RETSAM* to perform retiming of synchronous circuits and 2) Algorithm *CHECKCP* to check the feasibility of a specific clock period. These two algorithms are explained in the following subsections.

6.1 *RETSAM*: Algorithm for Retiming

Retiming a synchronous circuit is achieved by performing a binary search of all possible clock periods of the circuit. The pseudo-code of the retiming algorithm is shown in Figure 2. 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).

6.2 *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. Pseudo-code for the algorithm that determines the feasibility of a clock period is shown in Figure 3. Lag ranges are stored in an array called $r[]$. The timing constraints are derived from the SAM. An approximate solution can be obtained for the minimum clock period if the clock period feasibility test *CHECKCP* is terminated once the binary search bounds become sufficiently tight.

The most important step in *CHECKCP* is solving for the vertex lags, $r()$. The objective of the retiming algorithm is to yield a set of vertex lags that satisfy (15) through (19). To achieve this, the vertex lag ranges are initialized with unbounded values ($[-\infty \ldots \infty]$). Timing constraints are continuously applied to these vertex lags in order to tighten

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 $c = CP_{target}$
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 $CP_{min} = CP_{max}$

Figure 2: Pseudo-code for *RETSAM*

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). Table 2 exemplifies the solution method for the vertex lags of the graph shown in Figure 1.

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

$$r(v_a) - r(v_b) = k, \tag{20}$$

$$r(v_a) - r(v_b) \leq k, \tag{21}$$

$$r(v_a) - r(v_b) = k_1 \ \ or \ \ r(v_c) - r(v_d) = k_2, \tag{22}$$

$$r(v_a) - r(v_b) = k_1 \ or \ r(v_c) - r(v_d) \leq k_2, \tag{23}$$

$$\begin{aligned} r(v_a) - r(v_b) = k_1 \ \ or \ \ r(v_c) - r(v_d) = k_2 \\ or \ \ r(v_e) - r(v_f) \leq k_3, \end{aligned} \tag{24}$$

where $r()$ are vertex lags and k_n are constant values. The *or* statements that appear in (22), (23), and (24) prohibit the use of standard linear programming methods [15] and necessitate the use of branch and bound techniques. Note the existence of *multiple choices* in each inequality.

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

Figure 3: Pseudo-code for clock period feasibility test, CHECKCP

To gain insight into how these multiple choice inequalities are created, consider retiming the graph of Figure 1, for which the SAM is shown in Table 1. To achieve a clock period of $c = 23$ tu, the dark shaded and light shaded paths must be avoided, since they represent long paths for $c = 23$ 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,

$$r(3) - r(4) = 1 \ or \ r(0) - r(1) = 1 \ or \ r(4) - r(0) \leq -1, \tag{25}$$

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 $p : e_3 \rightsquigarrow e_0$ non-existent, or at least one register must be placed between the initial and terminating vertices of the path p.

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 1. In this algorithm, the unbounded value $[-\infty \ldots \infty]$ is initially assigned to each vertex lag range. Only one vertex lag ($r(0)$ for simplicity) is initialized to 0 and the other lags are calculated relative to $r(0)$. The vertex lag non-negativity constraint from (15) is applied to each vertex, shown in the first three rows of Table 2, thereby further tightening the vertex lag ranges. It is shown in row 4 that the constraint from (15) cannot be used to further tighten the vertex lag ranges. The long path constraints from (16), (17), and (18) are therefore used in row 4 to further tighten the bounds. 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 non-negative edge weights on each edge. The algorithm may reach a point where the application of the constraints can no longer tighten the bounds any further. This situation is indicated in Table 2 by a dark shaded row, where the application of each of the long path, short path, and non-negativity rules cannot tighten any further the vertex lag ranges. Once this occurs, all possible values for each vertex lag are tested. On the first dark shaded row in Table 2, 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.

Table 2: Example solution for $c = 23$. 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	$-\infty..\infty$	$-\infty..\infty$	$-\infty..\infty$	$-\infty..\infty$	$-\infty..\infty$	$-\infty..\infty$	$-\infty..\infty$
r(0)-r(1)≤1	Negativity on e_0	0	$-1..\infty$	$-\infty..\infty$	$-\infty..\infty$	$-\infty..\infty$	$-\infty..\infty$	$-\infty..\infty$	$-\infty..\infty$
r(1)-r(7)≤0	Negativity on e_7	0	$-1..\infty$	$-\infty..\infty$	$-\infty..\infty$	$-\infty..\infty$	$-\infty..\infty$	$-\infty..\infty$	$-1..\infty$
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
⋮									

7 Experimental Results

The retiming algorithm *RETSAM* is implemented in C on a SUN 4 workstation. To permit evaluating the proposed retiming algorithm, modified MCNC benchmark circuits [16, 17] have been analyzed using this algorithm and compared with an implementation of the Leiserson-Saxe retiming algorithm [1]. The resulting clock periods of the retimed benchmark circuits are reported.

The application of RETSAM to the example MCNC benchmark circuits are described in Table 3. The initial five 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 and 3) vertices in the graph of each circuit, 4) the latency of the circuit, and 5) the original clock period. The sixth column contains the minimum clock period of the retimed circuit using RETSAM. The final column lists the clock period of the benchmark circuits that were retimed using standard retiming algorithms without considering RECs.

In these circuits, the average register delay ($T_{C \rightarrow Q} + T_{Set-up}$) 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. As shown in Table 3, the minimum clock period of the majority circuit from LGSynth89 derived from RETSAM is less than from existing retiming algorithms. This occurs since localized negative clock skew [5, 12] 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 are exant in those circuits retimed by RETSAM, a conclusion that cannot be drawn with other retiming algorithms.

8 Conclusions

A retiming algorithm is presented which considers variable clock distribution, register, and interconnect delay. To permit the consideration of these delay components, register electrical characteristics (RECs) are attached to each edge 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 the determination of the minimum clock period of the retimed synchronous circuit. An iterative method using ranges of vertex lags rather than constant vertex lags is presented to solve for the edge weights.

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

Summarizing, modified MCNC benchmark circuits have been retimed using a new algorithm which considers the effects of variable clock distribution, register, and interconnect delay on local data paths. This algorithm represents a significant generalization of existing retiming algorithms, permitting the use of retiming for the automated synthesis of higher speed, more reliable pipelined digital systems.

Table 3: 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)
	Edges	Vertices	Latency	Initial T_{CP} (tu)		
LGSynth89 - multi-level (netblif)						
C17	26	19	6	92	29	25
b1	34	19	5	80	33	28
cm138a	62	29	4	110	43	38
cm42a	65	34	3	101	46	38
cm82a	59	37	4	139	47	40
majority	26	17	5	103	33	35
LGSynth91 - multi level (blif)						
C17	19	12	5	63	32	26
b1	16	10	2	50	33	24
cm42a	49	18	4	78	35	32
cm82a	22	12	3	49	28	26
cm85a	70	36	3	102	51	40
cm150a	69	38	2	90	56	48
cm151a	38	22	3	93	41	36
decod	89	24	4	69	43	37
parity	47	32	2	77	49	36
tcon	65	34	2	40	34	24
Figure 1 with the added REC values						
ls91	11	8	4	45*	23	19

* denotes a graph that has race conditions before retiming

References

[1] C. E. Leiserson and J. B. Saxe, "Retiming Synchronous Circuitry," *Algorithmica*, Vol. 6, pp. 5–35, January 1991.

[2] A. T. Ishii, C. E. Leiserson, and M. C. Papaefthymiou, "Optimizing Two-Phase, Level-Clocked Circuitry," *Proceedings of the 1992 Brown/MIT Conference on Advanced Research in VLSI and Parallel Systems*, pp. 245–264, March 1992.

[3] G. De Micheli, "Synchronous Logic Synthesis: Algorithms for Cycle-Time Minimization," *IEEE Transactions on Computer-Aided Design*, Vol. CAD-10, No. 1, pp. 63–73, January 1991.

[4] J. P. Fishburn, "Clock Skew Optimization," *IEEE Transactions on Computers*, 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," *Proceedings of the IEEE Asia-Pacific Conference on Circuits and Systems*, pp. 12–17, December 1992.

[6] T. Soyata, E. G. Friedman, and J. H. Mulligan, Jr., "Integration of Clock Skew and Register Delays into a Retiming Algorithm," *Proceedings of the IEEE International Symposium on Circuits and Systems*, pp. 1483–1486, May 1993.

[7] B. Lockyear and C. Ebeling, "The Practical Application of Retiming to the Design of High-Performance Systems," *Proceedings of the IEEE International Conference on Computer-Aided Design*, pp. 288–295, November 1993.

[8] T. Soyata and E. G. Friedman, "Synchronous Performance and Reliability Improvement in Pipelined ASICs," *Proceedings of the ASIC Conference*, September 1994.

[9] S. Simon, E. Bernard, M. Sauer, and J. A. Nossek, "A New Retiming Algorithm for Circuit Design," *Proceedings of the IEEE International Symposium on Circuits and Systems*, pp. 4.35–4.38, May/June 1994.

[10] L. Chao and E. H. Sha, "Retiming and Clock Skew for Synchronous Systems," *Proceedings of the IEEE International Symposium on Circuits and Systems*, pp. 1.283–1.286, May/June 1994.

[11] E. G. Friedman and J. H. Mulligan, "Clock Frequency and Latency in Synchronous Digital Systems," *IEEE Transactions on Signal Processing*, Vol. 39, No. 4, pp. 930–934, April 1991.

[12] E. G. Friedman, "Clock Distribution Design in VLSI Circuits — an Overview," *Proceedings of the IEEE International Symposium on Circuits and Systems*, pp. 1475–1478, May 1993.

[13] K. A. Sakallah, T. N. Mudge, T. M. Burks, and E. S. Davidson, "Synchronization of Pipelines," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, Vol. CAD–12, No. 8, pp. 1132–1146, August 1993.

[14] S. Malik, E. M. Sentovich, R. K. Brayton, and A. Sangiovanni-Vincentelli, "Retiming and Resynthesis: Optimizing Sequential Networks with Combinatorial Techniques," *IEEE Transactions on Computer-Aided Design*, Vol. CAD-10, No. 1, pp. 74–84, January 1991.

[15] E. L. Lawler, *Combinatorial Optimization: Networks and Matroids*. Holt, Rinehart and Winston, NewYork, 1976.

[16] R. Lisanke, "Logic Synthesis and Optimization Benchmarks User Guide: Version 2.0," Tech. Rep., Microelectronics Center of North Carolina, December 1988.

[17] S. Yang, "Logic Synthesis and Optimization Benchmarks User Guide: Version 3.0," Tech. Rep., Microelectronics Center of North Carolina, January 1991.