Synchronous Performance and Reliability Improvement in Pipelined ASICs

Tolga Soyata and Eby G. Friedman

Department of Electrical Engineering
University of Rochester
Rochester, NY 14627

Abstract — The clock frequency of a synchronous circuit can be increased at the expense of increased system latency, area, and power using synchronous optimization techniques such as pipelining and retiming. Pipelining is a well developed methodology, having been applied to almost every computer architecture from microprocessors to supercomputers. Retiming, on the other hand, has only recently become popular and practical application areas are currently being developed. Both pipelining and retiming are reviewed in this paper.

In order to make retiming more generally useful, low-level circuit delay components inherent to ICs must be incorporated into the retiming process. These issues include variable register delay, clock skew, and interconnect delay. An algorithm is presented by the authors for incorporating variable register delays, interconnect delay, and clock skew into retiming. This algorithm identifies and eliminates path-dependant race conditions in synchronous circuits. The results of applying the algorithm to MCNC benchmarks is presented and both performance and reliability improvements are observed.

I. INTRODUCTION

The performance of synchronous ASICs can be increased by pipelining at the expense of increased system latency and area. Pipelining converts a combinatorial circuit into its sequential equivalent by breaking the global data paths into local data paths with smaller delay. This is achieved by inserting registers (memory elements) between logic blocks. The intermediate processed results are saved in these memory elements and used during the following clock cycles [1]. Thus, this technique increases the rate of data flow by providing concurrent operation, albeit, with increased circuit area and system latency.

Retiming is a technique used to increase the clock frequency in pipelined synchronous circuits without affecting synchronous latency. An initial synchronous system is converted via retiming into a functionally equivalent system using techniques originally described by Leiserson and Saxe [2]. The locations of the registers are changed so as to minimize the clock period while preserving the system function and latency. The primary distinction between pipelining and retiming is that pipelining converts a combinational circuit into a sequential one, increasing system latency. In retiming, alternatively, the register locations within a sequential circuit are optimized such that the circuit operates at the highest possible frequency without increasing the latency.

Even though retiming can achieve a lower clock period, currently proposed retiming techniques do not incorporate practical circuit issues such as variable register delay, clock skew, and interconnect delay. By incorporating these circuit issues into retiming, a more accurate and reliable estimation of the register locations can be determined. Furthermore, catastrophic race conditions are eliminated as part of the retiming process.

A detailed review of both pipelining and retiming is provided herein. A new retiming algorithm that integrates the aforementioned practical circuit delay components is also presented. The paper is organized as follows. A discussion of pipelining techniques and related work in the field is provided in Section II. Retiming of synchronous circuits and related retiming algorithms are discussed in Section III. In Section IV, modeling of path dependent clock distribution, register, and interconnect delays is presented for use in the retiming algorithm described in this paper. The algorithm utilizes these delays to more accurately and reliably optimize the register locations within a pipelined system and is demonstrated using modified MCNC benchmark circuits. Finally, some conclusions are drawn in Section V.

II. PIPELINING OF SYNCHRONOUS DIGITAL SYSTEMS

Pipelining is an effective technique for increasing the performance of synchronous circuits. After pipelining a combinatorial circuit, the clock frequency of the circuit is increased, resulting in higher performance. In subsection A, a detailed description of the terms pertaining to pipelining are reviewed. Previous work in the field is summarized in subsection B, emphasizing speed, area, and efficiency trade-offs in synchronous pipelined systems.

A. Definitions related to Synchronous Pipelining

In this section, common terms used in the pipelining literature are defined. This discussion is intended to serve as a guide for the remainder of the paper.

The **latency** of a circuit is vaguely defined within the literature. For a combinatorial circuit, the latency of the circuit is defined as the *time* required for the signal to reach the system output after arriving at the system input. This definition has no ambiguity. For synchronous systems, however, the definition must be extended. For a synchronous system, two different latencies may be defined: **temporal latency** is the time required for a signal to reach the system output after arriving at the system input, whereas **sequential latency** is the *number of clock periods* required for the data signals to reach the system output upon arriving at the system input. The difference between these two definitions is significant since often sequential optimization operations do not change the sequential latency, but do change the temporal latency. These definitions of latency are used throughout this paper in order to provide insight into different effects of sequential optimization on system latency.

The **clock frequency** of a sequential circuit is the rate at which new data flow into the system and appear at the output. The primary goal of sequential optimization is to increase the clock frequency or decrease the **clock period**, which is the reciprocal of the clock frequency. The relationship between clock period and latency is further investigated in [3, 4].

Due to the nature of clock distribution networks in sequential circuits [5], differences in delay are created between the arrival times of the clock signals at different registers. The absolute delay of the clock signal from the clock source to a specific node is the **clock delay** and the differences between the clock delays of any two registers is the **clock skew** between these nodes. The notion of **localized clock skew** and its application to increasing the

0-7803-2020-4/94 $4.00 © IEEE

clock frequency within pipelines is introduced by Friedman and Mulligan in [3]. 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 important in pipelined systems since non-sequentially adjacent registers do not receive data at adjacent clock intervals, and therefore the clock skew between them is unimportant.

Race conditions are caused by *early-clocking*, i.e., clocking of registers before the relevant data is successfully latched. **Negative clock skew** occurs if the initial clock signal leads the final clock signal of a local data path. A race condition occurs if the skew is negative and greater in magnitude than the total local data path delay [5, 6]. Those paths with negative delay are called **short paths** [7]. Similarly, a **long path** designates those paths with a delay greater than the desired clock period of the circuit.

B. Previous Work in the Field of Pipelining

Pipelining has been used to improve the speed of a number of different applications, ranging from combinatorial circuits to microprocessors and DSP-based systems. This section is divided into four subsections: early work in the field of pipelining combinatorial circuits is reviewed in the first subsection. Pipelining of microprocessors and DSPs is discussed in the following two subsections, respectively, followed by a brief review of wave-pipelining in the last subsection.

B.1 Pipelining of Combinatorial Circuits

One of the earliest studies of pipelining was by Cotten in 1965 [1] in which he describes the time required for a data signal to reach the system output once it is applied to the system input as the **pipeline fill-up time**, and the rate at which the data flow in the pipeline as the **byte-flow**. Figure 1 depicts a pipelined circuit in which the registers are placed between logic elements so as to increase the data flow rate.

The dependance of the maximum flow-rate on the register delays was further investigated by Cotten in [8] and others [3, 4]. Their work showed that due to the inherent delay of the pipeline registers, the computational speed cannot be increased arbitrarily, but rather is bounded by the register delays. Jump and Ahuja [9] assign costs to registers and the logic elements and study the average delay, the average cost/operation, and the average time/operation ratios in a quantitative framework. In this paper, the *delay* of the circuit δ is defined as

$$\delta = N(T_S + T_R), \tag{1}$$

where N is the number of pipeline stages in the circuit, and T_S and T_R are the maximum logic and register delays, respectively. This definition corresponds to that of the temporal latency introduced earlier. Note that Jump and Ahuja assume uniform delays for each stage and only the maximum delay is considered. In the same paper, the pipeline efficiency is analyzed using measures such as the average cost per operation $\eta(M)$ and the average time per operation $\tau(M)$. These ratios are defined as

$$\eta(M) = (K_P + K_R N_R)(T_S + T_R)\left(\frac{M + N - 1}{M}\right), \tag{2}$$

$$\tau(M) = (T_S + T_R)\left(\frac{M + N - 1}{M}\right), \tag{3}$$

respectively, where K_P and K_R are the total cost of the logic elements and a single flip-flop per second, respectively, M is the average number of operations to be performed, and N_R is the total number of stages. In this context the definition of the *cost* is left open-ended, i.e., parameters such as power consumption or area can be used as a measure of the cost depending upon the application. They further show that as the number of operations increase, the term $(M + N - 1)/M$ approaches unity. This term is defined earlier as the **efficiency** of a pipeline by Chen [10] and can be interpreted as N clock periods are required to perform the initial operation and the remaining $M-1$ operations occur at each following clock period. As the number of operations increase, the performance degradation due to the pipeline fill-up time becomes less significant.

Another specific application of pipelining to combinatorial circuits is arithmetic functions, investigated by Hallin and Flynn [11]. They define the efficiency of pipelining as

$$pipeline\ efficiency = \frac{N}{DG}, \tag{4}$$

where N is the number of bits in the operands, D is the delay of each pipeline stage (assumed uniform), and G is the total number of gates in the total system including the latches. A wide range of adders (e.g., carry look-ahead, conditional-sum) and multipliers (e.g., Wallace, fully iterative array) are contrasted. They show that as the pipeline depth is increased by a factor k, the efficiency does not increase by the same factor due to the added overhead of the registers.

Global data paths are broken up into local data paths so as to achieve a specified clock period. Papaefthymiou presents an algorithm in [12] for automating pipelining of a fully combinatorial circuit in $O(E)$ time.

All of the previously discussed papers assume the pipelines consist of edge-triggered latches only. In [7], Sakallah *et al.* describe synchronizing pipelines consisting of multi-phase latches. A previously introduced timing model [13] is applied and both short and long path constraints are introduced.

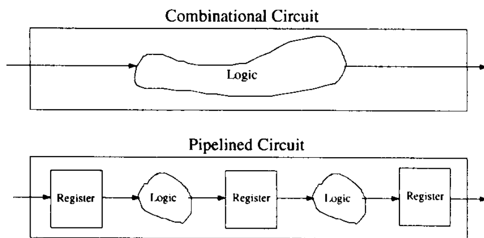

Figure 1: Pipelining breaks global data paths into local data paths with smaller delay so as to increase the data flow rate.

B.2 Pipelining of Microprocessors

The application of pipelining to processor design by parallelizing the fetch, decode, and execute units was initially studied by Flynn in 1966 [14]. He showed that by parallelizing the events in a SISD (single instruction, single data) machine, it is possible to increase the rate at which the input of the system accepts new data and the rate at which the system outputs processed data. Figure 2 conceptualizes pipelining of the DLX processor [15] in which

instruction fetch (F), instruction decode (D), execute (X), memory access (M) and writeback (W) operations are performed in parallel. Although these five operations are necessary to complete an instruction, each instruction effectively requires a single cycle due to the inherent parallelization. Note the pipeline stall (denoted as "s") which occurs at the fetch phase of the sixth instruction. Due to this hazard, the pipeline is not fully utilized in the sixth clock cycle. The shaded column denotes the clock cycle in which there are no pipeline hazards, resulting in a 100% pipeline utilization.

Although pipelining can increase synchronous operation dramatically, it cannot be fully exploited in microprocessor architectures due to instruction dependencies, structural limitations, and branch instructions. Complete parallelization of the code is not possible since some instructions need operands produced by previous instructions [16, 17]. Branch delay and branch prediction methods have been employed to overcome this problem [15, 18, 19]. Those deficiencies that decrease pipeline efficiency are called **hazards** and cause pipeline **stalls** (situation where the execution of an instruction must be delayed due to a hazard).

Pipelining is widely used in supercomputers. The relationship between the degree of central processor pipelining and supercomputer performance is discussed by Kunkel and Smith [20]. They show that overall pipeline performance peaks at six gates per pipeline segment. Using excessive gates per segment degrades the performance since the clock period is increased. Pipeline segments that use too few gates degrade the performance due to data and clock skews within the system. Note that **data skew** is defined as the difference in delay between the maximum and minimum signal propagation times through the combinational logic within the pipeline stages.

B.3 Pipelining of DSPs

The application of pipelining to enhance the performance of digital signal processors (DSPs) has been well studied [e.g., 21–23]. Capello and Steiglitz define **completely-pipelined architectures** in [22] in which circuits are pipelined down to the bit level. They apply pipelining to DSP architectures and show that complete pipelining is appropriate for array-connected (mesh-connected) DSP architectures. Capello, LaPaugh, and Steiglitz define an AP-product to measure the efficiency of pipelining, where A is the area of the VLSI chip and P is the clock period. This definition of efficiency is similar to Hallin and Flynn's definition in that the term G has similarities to A (since the number of gates used in the circuit is directly proportional to the chip area) and D is similar to P. Siomalas and Bowen investigate [23] strategies for designing DSPs such that all building blocks within the DSP are active at the same time. This methodology increases the effective speed of the DSP operations since the most efficient use of pipelining is achieved by maximally exploiting the inherent temporal parallelism. They also demonstrate their method of pipelining on FFT design.

Instruction	Clock Cycle						
	1	2	3	4	5	6	7
1	F	D	X	M	W		
2		F	D	X	M	W	
3			F	D	X	M	W
4				F	D	X	M
5					F	D	X
6						s	F

Figure 2: Pipelining of microprocessors: five primary operations of the microprocessor are pipelined to increase computational speed.

B.4 Wave-Pipelining

The clock frequency of a pipelined synchronous system can be increased without increasing the number of registers. This is possible by applying input signals faster than the total delay of the data path.

Successive *waves* of data are sent through combinatorial logic paths. If the data skew is small and sufficiently accurate control of the arrival times at every node is maintained, the successive data waves can act as a pipeline, permitting fewer synchronizing registers to be used. This technique is called **wave-pipelining** and was originally proposed by Cotten in 1969 [8]. Figure 3 depicts a wave-pipelined circuit in which successive data waves propagate through the pipeline.

Although wave-pipelining can be successfully applied to highly-structured architectures, it fails to work well in those architectures in which the path delays are unbalanced (significant data skew exists). The insertion of active elements in the system to permit wave-pipelining of unstructured architectures has been investigated by Wong, De Micheli, and Flynn [24]. The primary issue in achieving wave-pipelining is to equalize the path delays within the circuit by **padding** delays. Padding is the process in which those paths that have shorter delays are detected, permitting the insertion of active delay elements along these paths to ensure that all circuit paths have a delay between a lower and an upper bound. Recently, Shenoy *et al.* proposed greedy heuristic algorithms for padding these unequal delay paths [25]. They consider the padding operation as a post-processing step and offer linear programs for solving this problem. Wong, De Micheli, and Flynn also propose algorithms to implement wave-pipelining by padding delays [26]. They demonstrate padding on a 63–bit population counter and show that the clock frequency of the circuit can be increased by a factor of 2–3× using wave-pipelining. They also show that bipolar technologies, such as CML and super-buffered ECL, are more suitable for wave-pipelining than CMOS since they have inherent uniform delay (i.e., are less sensitive to input waveform and output loading).

III. RETIMING TECHNIQUES FOR SEQUENTIAL CIRCUIT PERFORMANCE OPTIMIZATION

Retiming is a sequential optimization technique used to increase the operating frequency of a synchronous circuit without increasing the sequential latency of the circuit. The

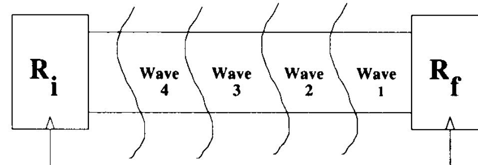

Figure 3: In wave-pipelining, successive data waves propagate through the logic elements, forming an effective pipeline.

location of the pipeline registers are reorganized so as to achieve the minimum clock period of a synchronous circuit while preserving the latency and the function. The relative timing of the internal events may change, however, the overall behavior of the circuit is preserved. In subsection A, the original paper on retiming published by Leiserson and Saxe is reviewed and a survey of more recent work in the field of retiming is presented in subsection B.

A. Overview of the Retiming Process

Leiserson and Saxe originally showed in 1981 [27] that it is possible to obtain a functionally equivalent sequential circuit that operates faster by changing the locations of the registers according to a set of rules. A methodology for optimally determining the location of these registers such that the minimum clock period is achieved while retaining the system latency and functionality is known as **retiming** [27] and is recently reviewed in [2].

In retiming, a circuit description containing logic delays and path connections is transformed into a directed graph in which the vertices represent logic delays and the edges between these vertices represent connections between these logic elements. Weights are assigned to each edge, defining the number of registers between logic elements. A zero weight edge, for example, shows that no registers exist between those two vertices. Edges are assumed to have no effect on the path delays and each data path is defined from vertex to vertex. In the Leiserson-Saxe algorithm [2], the edge weights are varied such that the function and latency of the original system is preserved, while tracking the effective latency at each vertex. The location of these registers are constrained by specific retiming rules, which ensure functional correctness and equivalency. Note that in minimizing the clock period, a successive search is performed to determine the minimum feasible clock period rather than choosing a register allocation that satisfies a specified clock period. The original sequential circuit from [2] is shown in Figure 4a along with a retimed version which is depicted in Figure 4b. The minimum clock period of the original circuit is 24 **time units** (tu) and the retimed version has a clock period of 13 tu. The clock period is decreased by changing the register locations such that data paths with large delays are broken into smaller paths with less delay. This process reduces the delay of the limiting critical path which constrains the overall system clock period. The critical paths in the original and retimed versions of the circuit are shown in bold in Figure 4.

An important step in the retiming process is producing a set of edge weights that satisfy a specific set of constraints. The process of solving for a set of feasible edge weights requires a solution for $|E|$ unknowns, where $|E|$ is the number of edges in the sequential circuit. A key aspect of retiming, originally described in [27], is the use of **vertex lags** to reduce the number of unknowns from $|E|$ to $|V|$ based on the observation that, to preserve functional equivalency, edge weight changes cannot be made independently. An integer lag is assigned to each vertex and the registers are moved from edge to edge, changing the vertex lags according to

$$w_r(e) = w(e) + r(u) - r(v), \tag{5}$$

where $w_r(e)$ and $w(e)$ are the weights of edge e after and before retiming, respectively, and $r(u)$ and $r(v)$ are the lags of vertices u and v, which are connected to the back and front of edge e, respectively [2].

The retiming process is based on (5) and two primary rules: 1) edge non-negativity constraints and 2) long path constraints. These two rules generate inequalities in terms of vertex lags. The set of linear inequalities are solved using the Bellman-Ford algorithm [28] and the resulting lags are used to calculate the retimed edge weights, $w_r(e)$, from (5). The two aforementioned retiming rules are repeated here: the edge non-negativity constraint is

$$r(u) - r(v) \leq w(e), \; \forall e : u \rightarrow v, \tag{6}$$

which ensures non-negative weights on each edge. The long path constraint is

$$r(u) - r(v) \leq W(u,v) - 1, \; \forall u, v : \; D(u,v) > c, \tag{7}$$

where $W(u,v)$ and $D(u,v)$ are matrices containing the total weight and the total delay of the path from vertex u and vertex v, respectively, and c is the target clock period. The last rule states that a register must be placed on all paths with a delay greater than the desired delay, i.e., a path with an excessively large delay must be broken into smaller paths by placing a register in the middle. Note that in calculating the minimum clock period of the circuit, a binary search is performed and the minimum achievable clock period is selected as the clock period of the circuit.

B. Previous Work in the Field of Retiming

The field of retiming has developed to include register minimization and improved propagation delay models [2, 29]. A single value is assigned to each vertex and edge, representing the delay of the logic elements and the

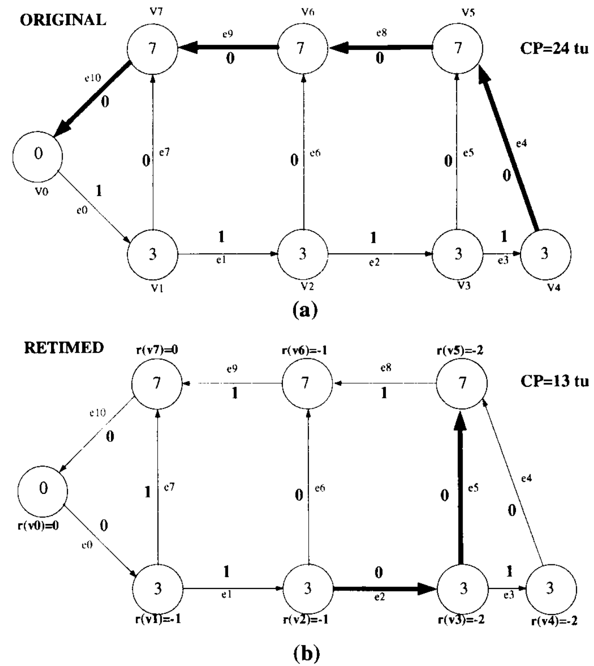

Figure 4: a) The original graph introduced in [2] and b) its retimed version. Note that the sequential latency is four clock periods in both cases. $r(v_n)$ denotes the lag of vertex n after retiming.

number of registers between the logic elements, respectively. Using this approach, the problem of clock period minimization for synchronous circuits can be solved using the aforementioned Bellman-Ford algorithm [28]. Similar ideas are used to perform minimum clock pipelining, minimum clock retiming, and approximate minimum clock retiming [12]. Papaefthymiou [30] shows that the total delay of the **cycles** (paths initiating and terminating at the same vertex) in a graph representing a synchronous circuit plays a significant role in defining the maximum clock frequency of a synchronous circuit. Upper and lower bounds for the clock period are derived using the edge weights and total delays of the cycles. The lower bound on the clock period is characterized as the maximum ratio of the delay to weight for any cycle in the graph. In Figure 4a, for example, there are four cycles with total delays (10,20,30,33) and total weights (1,2,3,4). Let C_0 through C_4 represent these cycles, where the lowest index represents the innermost cycle. Thus, C_0 has a total delay $d(C_0)$ and weight $w(C_0)$,

$$\begin{aligned} d(C_0 = v_0 \rightarrow e_0 \rightarrow v_1 \rightarrow e_7 \rightarrow v_7 \rightarrow e_{10}) &= 10, \\ w(C_0 = v_0 \rightarrow e_0 \rightarrow v_1 \rightarrow e_7 \rightarrow v_7 \rightarrow e_{10}) &= 1, \end{aligned} \quad (8)$$

where $d()$ and $w()$ are the delay and weight functions, respectively. Thus, the data must travel C_0 (10 tu) in 1 clock cycle, cycle two (20 tu) in 2 clock cycles, etc. The lower bound on the clock period of this circuit is therefore [30]

$$T_{min} = \max\left\{\frac{10}{1}, \frac{20}{2}, \frac{30}{3}, \frac{33}{4}\right\} = 10, \quad (9)$$

where T_{min} is the minimum achievable clock period of the circuit. Note that this clock period can only be achieved when the delays are properly distributed. In a circuit in which the delays are non-uniform, this minimum limit is not achievable. For example, the retimed version of Figure 4a has a clock period of 13 tu, whereas the theoretical minimum is 10 tu.

Ishii and Leiserson [31] and Sakallah, Mudge and Olukotun [13] develop a theory for analyzing level-clocked circuitry. In [31], an algorithm is presented for verifying timing in VLSI circuits. Propagation delays of latches are considered to be constant and minimum propagation delays of the logic elements are not considered. Retiming algorithms for synchronous circuits consisting of single-phase [32], two-phase [33], and multi-phase [34] flip-flops have also been developed.

Retiming can be made more effective by combining it with combinational optimization. Algorithms have been proposed by DeMicheli [35] to minimize the cycle time using logic transformations, such as elimination, resubstitution, extraction, and decomposition, while also retiming the synchronous circuit. Another proposed method is to temporarily shift the registers to the periphery of the synchronous circuit, perform logic minimization on the purely combinatorial circuit, and return the registers to within the circuitry [36]. Although this methodology temporarily creates negative edge weights, violating the aforementioned retiming rules [see (6)], once the registers are replaced, the negative edge weights are eliminated. This method of performing logic optimization on the combinatorial circuit between the registers is defined as **resynthesis** [36, 37]. Application of this method to multi-phase pipelines is discussed in [38].

Incorporating clock skew into retiming was initially proposed in [39] and a retiming algorithm which included both clock skew and variable register delays was first introduced by the authors in [6]. In [6], delay component values are attached to each edge describing the delay characteristics of the registers as well as the clock distribution network. Following this work, the inclusion of clock skew and register delays into retiming was presented in [40] using the clock model introduced in [13].

Retiming to minimize the number of registers in a sequential circuit was proposed by Leiserson and Saxe and is shown to be equivalent to state minimization in FSMs [2]. Recently, retiming has been extended to cover gated-clocks and precharged circuit structures [41]. Retiming to decrease power dissipation by minimizing the switching activity within the synchronous circuit while preserving functional equivalency has also been demonstrated [42].

Although retiming can significantly reduce the clock period of a synchronous circuit, its usefulness is limited by the nature of the circuit structure. The application of retiming to highly-structured circuits such as FIR filters is studied in [43] by combining retiming with algebraic speed-up techniques. This method is based on the ERB (eliminating retiming bottlenecks) method introduced in [44] in which the computational structure of the original circuit is changed so as to enhance its ability to be retimed.

IV. INCORPORATION OF ELECTRICAL TIMING INFORMATION INTO RETIMING

In early work on retiming, it is assumed that the clock skew is zero, register delays are either zero or constant, and interconnect delay is negligible [2, 35]. In practical integrated circuits, however, these delay terms play a critical role in circuit operation and must be considered in order to accurately and reliably retime a synchronous ASIC. In this section, an algorithm is proposed to more accurately retime synchronous circuits by considering path dependant clock distribution, register, and interconnect delays. A discussion of how these electrical issues are modeled and incorporated into the retiming delay equations is provided in subsection A, while the algorithm is briefly reviewed in subsection B. The algorithm is demonstrated on MCNC benchmark circuits. These results are discussed in subsection C.

A. Modelling of the Delay Components

In order to consider the effects of clock skew, variable register delay, and interconnect delay, a number set, the **Register Electrical Characteristic** (REC), is assigned to each edge of the graph describing the ASIC in the following form: $T_{CD} : T_{Set-up}/T_{C \rightarrow Q}$ — T_{Int1}/T_{Int2} [6]. T_{CD} is the clock delay from the 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_{Int1} and T_{Int2} are the interconnect delays incurred when the data signal is propagating to (from) the logic elements, respectively. If, after the register relocation process, no registers exist on an edge e, the total interconnect delay along edge e is defined to be $T_{Int1} + T_{Int2}$, whereas if one or more registers are located on this edge after relocating the registers, the interconnect delay is separated into two different values: T_{Int1} is the delay from the originating vertex of edge e to the input of the first register of edge e, and T_{Int2} is the delay from the output of the last register of edge e to the terminating vertex of edge e. Figure 5 depicts a graph with attached

REC values. In the graph, vertices represent logic elements and the values inside the vertices represent the delays of these logic elements. Note that each edge is assigned an REC value in the aforementioned form.

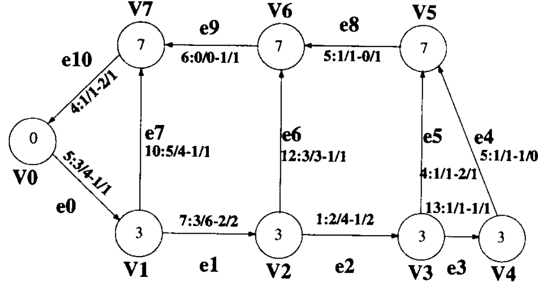

Figure 5: Graph of Figure 4 with attached REC values

An important step in the retiming process using this algorithm is the estimation of the REC values. Initial estimates of the REC values must be provided to the retiming algorithm. With these estimates, an initial exploratory retiming is performed. As the lower level information becomes better specified, the *exploratory* retiming process provides more accurate solutions. Therefore, the retiming process can be thought of as an iterative process in which low level information is continually fed back to the higher pipelining level until a satisfactory design is determined.

B. Retiming Algorithm which Considers RECs

In the proposed retiming algorithm, path delays are defined to be edge to edge [6] as opposed to vertex to vertex as in traditional retiming algorithms [2]. The delays between edges are calculated once and used throughout the retiming process. The path delay T_{PD} from edge e_i to edge e_j is the summation of the clock-to-Q delay ($T_{C \to Q}$), the interconnect delay ($T_{Int1} + T_{Int2}$), the logic delay (T_{Logic}), the set-up time (T_{Set-up}), and the local clock skew (T_{Skew}) between registers i and j, and is

$$T_{PD}(i,j) = T_{C \to Q}(i) + T_{Int2}(i) + T_{Logic}(i,j) + T_{Int1}(j) + T_{Set-up}(j) + T_{Skew}(i,j). \quad (10)$$

$T_{Skew}(i,j)$, the clock skew between edges i and j, is

$$T_{Skew}(i,j) = T_{CD}(i) - T_{CD}(j). \quad (11)$$

Note that if parallel paths exist between any two vertices, the concept of path delay $T_{PD}(i,j)$ is extended to $T_{PDmin}(i,j)$ and $T_{PDmax}(i,j)$, where $T_{PDmin}(i,j)$ and $T_{PDmax}(i,j)$ are the minimum and maximum path delays between edges i and j, respectively. A matrix called the Sequential Adjacency Matrix (or the S matrix) is introduced whose element $S(i,j)$ contains the path delay from edge e_i to edge e_j. This matrix is $E \times E$ in size, where E is the number of edges in the graph. The S matrix is also separated into two matrices, S_{min} and S_{max}, to evaluate the short and long path delays. The short path delays are used to detect race conditions and the long path delays are used to calculate the minimum clock period of the synchronous ASIC. The short paths are those paths that contain a negative path delay. Race conditions in the synchronous ASIC appear as negative entries in the S matrix.

The S matrix, which contains the delay of all possible data paths for any possible register placement, is created once for each circuit and latency. A binary search is then performed on the sorted values of the S matrix to decide which data path delays are achievable without creating any race conditions and/or paths with a delay larger than a target clock period. The binary search is terminated upon determining the minimum clock period of the ASIC. The register locations are calculated so as to realize this minimum clock period. Long path constraints are used to detect path delays that exceed the maximum allowed clock period. The long path constraints are defined as

$$S_{max}(i,j) > c, \quad (12)$$

where c is the maximum allowed clock period for the circuit and i and j are the indices for the initial and terminating edges of the long path, i.e., the path $p : e_i \rightsquigarrow e_j$ is not permitted if the path delay exceeds the desired clock period c.

The algorithm detects and removes race conditions by using short path constraints. In short paths, the data signal reaches the final register of that local data path before the final register is clocked, thereby causing the ASIC to function improperly. These short paths are created when the clock skew is negative and greater in magnitude than the path delays, creating a race condition [5, 39]. The short path constraints are defined as

$$S_{min}(i,j) \le 0, \quad (13)$$

implying paths with negative or zero delay. Negative entries in the S matrix create race conditions, since the entry represents a local data path with negative delay.

The addition of the short path constraint adds significant complexity and does not allow for simple modification of the original retiming algorithm introduced in [2]. Traditional methods which ignore negative clock skew produce inequalities in the form of

$$x_i - x_j \le a_{ij}, \quad (14)$$

whereas with the added short path constraints due to localized clock skew, the retiming algorithm presented in this paper produces inequalities in the form of

$$x_i - x_j \le a_{ij} \;\; or \;\; x_u - x_v \le a_{uv}, \quad (15)$$

where x_i, x_j, x_u and x_v are unknowns (used to define the locations of the registers in the algorithm) and a_{ij} and a_{uv} are constants. The addition of the boolean "*or*" statement is due to the introduction of the short path constraints and does not permit the use of standard linear programming techniques such as the aforementioned Bellman-Ford algorithm. To solve this set of inequalities, vertex lag *ranges* are used rather than vertex lags. Each vertex is assigned a lag range and the ranges are continuously tightened to satisfy all of the constraints for the specified target clock period. This process is continued until a minimum clock period is reached.

The S matrix of Figure 5 is shown in Table 1. Path delays greater than 23 tu are heavily shaded while path delays less than or equal to zero are lightly shaded. For T_{CP} = 23 tu, the heavily shaded elements in the table constitute the long paths and the lightly shaded elements represent the paths that contain race conditions. Other matrix entries indicate data paths with delays less than the target clock period of 23 tu and no existing race condition. The binary search process used in the algorithm evaluates each possible clock period until the minimum achievable clock period is determined.

Table 1: Sequential Adjacency Matrix for the graph of Figure 5. Light shaded entries represent short paths, whereas dark shaded entries represent long paths for $c = 23$. Unshaded entries denote permissible paths.

SAM (from \ to)	e0	e1	e2	e3	e4	e5	e6	e7	e8	e9	e10
e0		11	22	15			12	9			21
e1			20	13		23	10			22	
e2				-1	12	9			16		
e3					15				22		
e4									9	16	
e5									9	16	
e6										18	
e7											21
e8										9	22
e9	16	20					21	18			13
e10	5	9	20	13		23	10	7		22	19

C. MCNC Benchmark Application Results

The proposed retiming algorithm has been implemented in C on a SUN 4 workstation. To permit the evaluation of the enhanced retiming algorithm, modified MCNC benchmarks [45, 46] have been analyzed using this algorithm and compared with the Leiserson-Saxe retiming algorithm [2].

The application of the proposed retiming algorithm to the example MCNC benchmark circuits are described in Table 2. 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 [45, 46], 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 the new retiming algorithm. The final column lists the clock period of the benchmark circuits that was retimed using the standard Leiserson-Saxe retiming algorithms which do not consider RECs. In these circuits, the average register delay ($T_{C \to Q} + T_{Set-up}$) of the circuit is added to each local data path to compensate for the effects of the variable register delays. As shown in Table 2, the minimum clock period of the majority circuit from the LGSynth89 archive derived from the proposed retiming algorithm is less than from standard retiming algorithms. This occurs since localized negative clock skew [5, 39] subtracts delay from the critical path such that the worst case path has a smaller delay, thereby causing the minimum clock period to be less. Also, note that no race conditions are exant in those circuits retimed by the proposed algorithm, a conclusion that cannot be assumed with other retiming algorithms.

Table 2: Application of retiming to MCNC benchmark circuits with and without clock skew, interconnect, and variable register delays

Example	Graph properties: Edges	Vertices	Latency	Initial T_{CP} (tu)	T_{CP} After Retiming (tu)	T_{CP} $T_{SKEW}=0$ $T_{REG}=const$ (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

V. CONCLUSIONS AND FUTURE WORK

In this paper, synchronous circuit optimization techniques, such as pipelining and retiming, are reviewed and contrasted. Insight is provided into how these different approaches affect speed, area, power, and efficiency. Pipelining of microprocessors, DSPs, and wave-pipelining are briefly reviewed. The degradation of pipelining due to instruction dependencies and branch instructions in microprocessors is discussed and transformation techniques to prevent these inefficiencies are described.

Retiming techniques to optimally relocate the registers in a synchronous circuit are reviewed. Different approaches to retiming synchronous circuitry for minimizing the number of registers, the power dissipation, and the clock period are reviewed. Circuit types for which retiming are not applicable are mentioned and transformation techniques to permit retiming of these circuits are discussed.

A retiming algorithm for optimally relocating the registers of a synchronous pipelined ASIC which considers variable clock distribution, register, and interconnect delay is presented. To permit the consideration of these delay components, register electrical characteristics (RECs) are attached to each edge and the path delays are redefined to be from edge-to-edge.

The limitations and advantages of the retiming algorithm presented in this paper are compared using a set of modified MCNC benchmarks. The results of applying the algorithm to the benchmark circuits show that a more accurate and generalized relocation of the registers of a pipelined ASIC can be performed than with existing algorithms which do not consider clock distribution, register, and interconnect delay. Furthermore, the clock period can be further minimized due to localized negative clock skew. Also, catastrophic clock skew induced race conditions are detected and eliminated. Summarizing, this algorithm represents a significant generalization of existing retiming algorithms, permitting the accurate synthesis of higher speed and more reliable pipelined digital ASICs.

REFERENCES

[1] L. W. Cotten, "Circuit Implementation of High-Speed Pipeline Systems," *Proceedings of the AFIPS Fall Joint Computer Conference*, Vol. 27, pp. 489–504, November 1965.

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

[3] 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.

[4] E. G. Friedman and J. H. Mulligan, "Pipelining of High Performance Synchronous Digital Systems," *International Journal of Electronics*, Vol. 70, No. 5, pp. 917–935, May 1991.

[5] 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.

[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] 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.

[8] L. W. Cotten, "Maximum-Rate Pipeline Systems," *Proceedings of the Spring Joint Computer Conference*, Vol. 34, pp. 581-586, May 1969.

[9] J. R. Jump and S. R. Ajuha, "Effective Pipelining of Digital Systems," *IEEE Transactions on Computers*, Vol. C-27, No. 9, pp. 855-865, September 1978.

[10] T. C. Chen, "Parallelism, pipelining, and computer efficiency," *Computer Design*, Vol. 10, pp. 69–74, January 1971.

[11] T. G. Hallin and M. J. Flynn, "Pipelining of Arithmetic Functions," *IEEE Transactions on Computers*, pp. 880–886, August 1972.

[12] M. C. Papaefthymiou, "On Retiming Synchronous Circuitry and Mixed-Integer Optimization," Master's thesis, Massachussetts Institute of Technology, August 1990.

[13] K. A. Sakallah, T. N. Mudge, and O. A. Olukotun, "Analysis and Design of Latch-Controlled Synchronous Digital Circutis," *Proceedings of the 27th ACM/IEEE Design Automation Conference*, pp. 111-117, June 1990.

[14] M. J. Flynn, "Very High-Speed Computing Systems," *Proceedings of the IEEE*, Vol. 54, No. 12, pp. 1901–1909, December 1966.

[15] J. L. Henessy and D. A. Patterson, *Computer Architecture A Quantitative Approach*. Morgan Kaufmann Publishers Inc., 1990.

[16] G. S. Tjaden and M. J. Flynn, "Detection and Parallel Execution of Independent Instructions," *IEEE Transactions on Computers*, Vol. C-19, No. 10, pp. 889–895, October 1970.

[17] E. M. Riseman and C. C. Foster, "The Inhibition of Potential Parallelism by Conditional Jumps," *IEEE Transactions on Computers*, Vol. C-21, No. 12, pp. 1405–1411, December 1972.

[18] G. Kane, *MIPS RISC Architecture*. Prentice-Hall, 1988.

[19] INTEL Corporation, *The Intel Pentium^{TM} Processor. A Technical Overview*, 1994.

[20] S. R. Kunkel and J. E. Smith, "Optimal Pipelining in Supercomputers," *Proceedings of the 13th Annual Symposium on Computer Architecture*, pp. 404–411, 1986.

[21] N. R. Strader II, "VLSI Bit-Sequential Architectures for Digital Signal Processing," *Proceedings of the IEEE Conference on Acoustics, Speech and Signal Processing*, pp. 931–934, February 1987.

[22] P. R. Capello and K. Steiglitz, "Completely-Pipelined Architectures for Digital Signal Processing," *IEEE Transactions on Acoustics, Speech, and Signal Processing*, Vol. ASSP-31, No. 4, pp. 1016–1023, August 1983.

[23] K. O. Siomalas and B. A. Bowen, "Synthesis of Efficient Pipelined Architectures for Implementing DSP Operations," *IEEE Transactions on Acoustics, Speech, and Signal Processing*, Vol. ASSP-33, No. 6, pp. 1499–1508, December 1985.

[24] D. Wong, De Micheli, Giovanni, and M. Flynn, "Inserting Active Delay Elements to Achieve Wave Pipelining," *Proceedings of the IEEE Conference on Computer-Aided Design*, pp. 270–273, November 1989.

[25] N. V. Shenoy, R. K. Brayton, and A. Sangiovanni-Vincentelli, "Minimum Padding to Satisfy Short Path Constraints," *Proceedings of the IEEE International Conference on Computer-Aided Design*, pp. 156–161, November 1993.

[26] D. C. Wong, G. De Micheli, and M. J. Flynn, "Designing High-Performance Digital Circuits Using Wave Pipelining: Algorithms and Practical Experiences," *IEEE Transactions on Computer-Aided Design*, Vol. 12, No. 1, pp. 25–46, January 1993.

[27] C. E. Leiserson and J. B. Saxe, "Optimizing Synchronous Systems," *Proceedings of 22nd Annual Symposium on Foundations of Computer Science*, pp. 23–36, October 1981.

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

[29] C. E. Leiserson, F. M. Rose, and J. B. Saxe, "Optimizing Synchronous Circuitry by Retiming," *Proceedings of the 3rd Caltech Conference on VLSI*, pp. 87–116, March 1983.

[30] M. C. Papaefthymiou, "Understanding Retiming through Maximum Average-Weight Cycles," *Proceedings of the 3rd Annual Symposium on Parallel Algorithms and Architectures*, pp. 338–348, July 1991.

[31] A. T. Ishii and C. E. Leiserson, "A Timing Analysis of Level-Clocked Circuitry," *Proceedings of the Sixth MIT Conference on Advanced Research in VLSI*, pp. 113–130, March 1990.

[32] N. Shenoy, R. K. Brayton, and A. Sangiovanni-Vincentelli, "Retiming of Circuits with Single Phase Transparent Latches," *Proceedings of the IEEE International Conference on Computer Design*, pp. 86–89, October 1991.

[33] 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.

[34] B. Lockyear and C. Ebeling, "Optimal Retiming of Multi-Phase, Level-Clocked Circuits," *Proceedings of the 1992 Brown/MIT Conference on Advanced Research in VLSI and Parallel Systems*, pp. 265–280, March 1992.

[35] 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.

[36] 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.

[37] S. Malik, K. J. Singh, R. K. Brayton, and A. Sangiovanni-Vincentelli, "Performance Optimization of Pipelined Logic Circuits Using Peripheral Retiming and Resynthesis," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, Vol. CAD–12, No. 5, pp. 568–578, May 1993.

[38] N. V. Shenoy, R. K. Brayton, and A. Sangiovanni-Vincentelli, "Resynthesis of Multi-Phase Pipelines," *Proceedings of the 30th Design Automation Conference*, pp. 490–496, June 1993.

[39] 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.

[40] 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.

[41] A. T. Ishii, "Retiming Gated-Clocks and Precharged Circuit Structures," *Proceedings of the IEEE International Conference on Computer-Aided Design*, pp. 300–307, November 1993.

[42] J. Monteiro, S. Devadas, and A. Ghosh, "Retiming Sequential Circuits for Low Power," *Proceedings of the IEEE International Conference on Computer-Aided Design*, pp. 398–402, November 1993.

[43] Z. Iqbal, M. Potkonjak, S. Dey, and A. Parker, "Critical Path Minimization Using Retiming and Algebraic Speed-Up," *Proceedings of the 30th Design Automation Conference*, pp. 573–577, June 1993.

[44] S. Dey, M. Potkonjak, and S. G. Rothweiler, "Performance Optimization of Sequential Circuits by Eliminating Retiming Bottlenecks," *Proceedings of the IEEE International Conference on Computer-Aided Design*, pp. 504–509, November 1992.

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

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