Speeding up Pipelined Circuits through a Combination of Gate Sizing and Clock Skew Optimization

Harsha Sathyamurthy   Sachin S. Sapatnekar   John P. Fishburn

Abstract

An algorithm for unifying the techniques of gate sizing and clock skew optimization for acyclic pipelines is presented in this paper. In the design of circuits under very tight timing specifications, the area overhead of gate sizing can be considerable. The procedure described herein utilizes the idea of cycle-borrowing using clock skew optimization to relax the stringency of the timing specification on the critical stages of the pipeline. The theoretical basis for the procedure is developed, a new algorithm for timing analysis of acyclic pipeline circuits with deliberate skew is presented, and a sensitivity-based optimizer is used to solve the sizing+skew problem. Our experimental results verify that the procedure of cycle-borrowing using sizing+skew results in a better overall area-delay tradeoff as compared to using sizing alone.

1 Introduction

The problem of optimizing acyclic pipelines has attracted considerable interest of late; recent publications on the subject include [1, 2]. This paper presents a method for speeding up acyclic pipelined circuits through a combination of two techniques: gate sizing and clock skew optimization. Each of these techniques has been utilized in isolation for speeding up circuits; here, we present a method of unifying them to arrive at an optimal design. We begin with a brief introduction to each of these techniques.

1.1 Gate Sizing

The technique of gate sizing is well-known and several computer-aided design (CAD) tools (for example, [3–6]) have been developed to perform this optimization. The idea behind this optimization is simple. A typical digital integrated circuit consists of multiple stages of combinational logic blocks that lie between flip-flops (FF's) that are clocked by system clock signals. For correct circuit operation under a specified clock period, the worst-case input-output delay of each combinational stage must be restricted to be below a certain specification. Given the circuit topology, the delay of a combinational circuit can be controlled by varying the sizes of transistors in the circuit. In coarse terms, if we start from a minimum-sized circuit, the circuit delay can usually be reduced by increasing the sizes of certain gates in the circuit. Hence, making the circuit faster usually entails the penalty of increased circuit area and power dissipation. The trade-off involved here is, in essence, the gate sizing problem, and for the reasons specified above, it is typically applied to one combinational stage of a circuit at a time.

The problem of gate sizing is most commonly formulated as either

$$\begin{array}{ll} \text{minimize} & Area \\ \text{subject to} & Delay \leq P_{spec} \end{array} \qquad (1)$$

where P_{spec} is the specified clock period, or

$$\begin{array}{ll} \textit{minimize} & P \\ \textit{subject to} & Area \leq A_{spec} \end{array} \qquad (2)$$

where P is the clock period. In the former case, a goal period is specified, which translates into a restriction on the delay of each combinational segment, and in the latter formulation, the clock period is minimized subject to a specification, A_{spec}, on the area. *Area* is often measured as the sum of all the transistor channel widths.

1.2 Clock Skew Optimization

To understand the problem of clock skew optimization, it is important to recognize that due to differences in interconnect delays on the clock distribution network of integrated circuits, there is typically a skew between the arrival times of clock signals at the flip-flops (FF's). One approach ([7, 8] etc.) that has been followed by several researchers is to design the clock distribution network so as to ensure zero clock skew. An alternative approach [9,10] views clock skews as a manageable resource rather than a liability, and manipulates clock skews to advantage by intentionally introducing skews to improve the performance of the circuit. This process of selecting the optimal skews is the problem of clock skew optimization.

To illustrate the benefits of clock skew optimization, consider the following example. In Figure 1, if the combinational subcircuits CC_1 and CC_2 have delays of 6 and 14 units, respectively, then with zero skew, the fastest allowable clock has a period of 14 units. However, if the clock input to the FF B is skewed by -4 units relative to the other two FF's, then the circuit can run at a clock period of 10 units.

In the rest of this paper, we will use the word "skew" to mean the relative skew, with the reference set to the skews at the primary inputs (PI's) and primary inputs (PO's), each of which are zero.

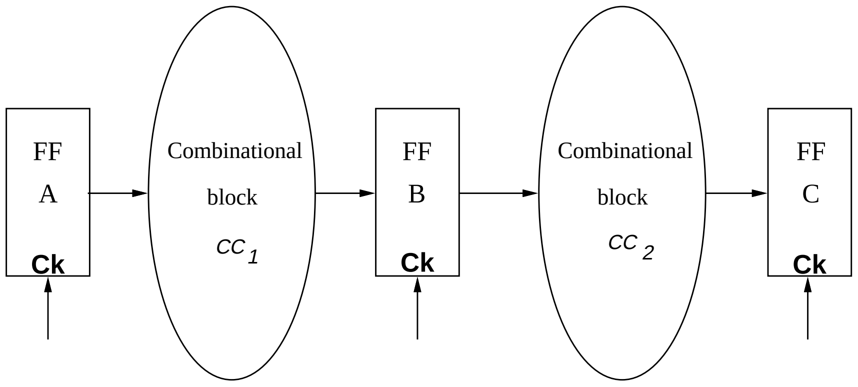

Figure 1: The advantages of nonzero clock skew.

This approach was formalized in the work by Fishburn [9] where clock skew optimization problem was formulated as a linear program (LP). An enhancement in [10] recognizes that the LP solution is degenerate, and proposes a method for choosing the optimal values for the nonbasic variables using criteria that incorporate the complexity of designing the clocking network.

1.3 Outline of the Paper

We begin by motivating the need for using skew optimization in conjunction with sizing. In the example circuit in Figure 1, it has been shown that a clock period of 10 units is possible using skew optimization alone by the application applying a skew of -4 units at FF B. To achieve this period using sizing alone would probably require a large increase in the area of subcircuit CC_2. For clock periods smaller than 10 units, the use of sizing is essential, and skew optimization alone will not be sufficient.

The general character of the area-delay tradeoff for sizing alone (without using skew optimization) is shown by the curve in Figure 2; this shows that the area increases nonlinearly with a decrease in the delay specification. For a loose delay specification, the area penalty is not very large, but for tighter specifications, it becomes extremely large. The use of skew optimization in conjunction with sizing would allow CC_2 to *steal* a part of the clock cycle for CC_1, thereby allowing it a larger timing budget and a correspondingly smaller area penalty. For example, in the figure, CC_2 may move from position A to position B. Correspondingly, CC_1 may move from position C to position D. The nonlinearity of the area-delay tradeoff curve ensures that the area corresponding to moving CC_1 from C to D is smaller than the area savings corresponding to moving CC_2 from A to B.

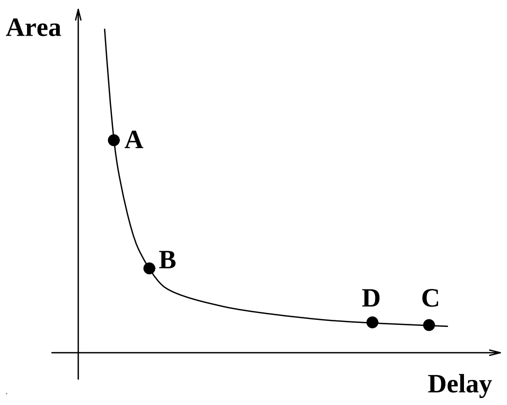

Figure 2: Using sizing in conjunction with clock skew.

In this paper, we present an algorithm for unifying clock skew optimization and transistor sizing. The algorithm is directed towards circuits with edge-triggered flip-flops. The solution is arrived at in two steps:

Step 1 The circuit delay is minimized and the long path constraints satisfied, ignoring the short path constraints altogether.

Step 2 The short path constraint violations are resolved by padding the circuit with buffers as necessary.

The above techniques are illustrated on single-phase clocked circuits containing edge-triggered flip-flops.

It was previously thought that it was extremely hard to achieve a good solution to this problem. In [9], this problem was shown to be a signomial programming problem, which is known to be difficult. To the best of of our knowledge, this is the first piece of work that shows that Step 1 corresponds to a convex optimization problem, which implies that it is easy to find a solution to this subproblem. Therefore, if the number of short path violations in the final circuit is relatively small, Step 2 will perturb the solution by only a small amount, and the above technique will work well for practical circuits. In this work, we concentrate on the solution to Step 1; Step 2 can be solved by variations of methods like [11]. Another contribution of this work is in the application of PERT to acyclic sequential circuits to detect long path constraint violations.

The organization of this paper is as follows. Section 2 formally states the problem to be solved, followed by an overview of the algorithm in Section 3. The procedure for detecting long path constraint violations is described in Section 4, and an account of how delay sensitivities for each gate are calculated is given in Section 5. The entire algorithm for Step 1 is then presented in Section 6. Finally, experimental results and concluding remarks are provided in Sections 7 and 8, respectively.

2 Statement of the Clock Skew Optimization Problem

2.1 Long Path and Short Path Constraints

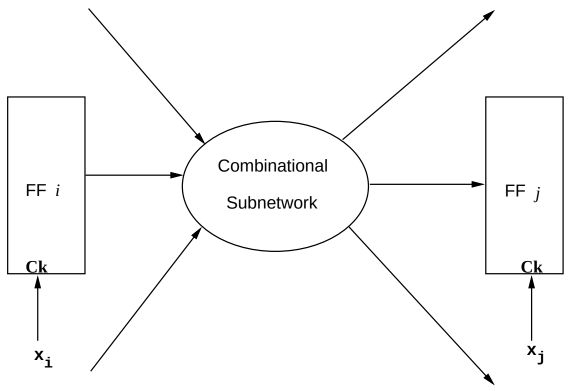

Figure 3: The clock skew optimization problem.

For each flip-flop pair (FF_i,FF_j), let x_i, x_j be the skews at the flip-flops FF_i and FF_j respectively and $d(i,j)$ be the delay of the combinational block between them (with $\underline{d}(i,j)$ being the minimum delay and $\overline{d}(i,j)$ being the maximum delay). This is illustrated in Figure 3 for the clock skew optimization problem. Let T_{hold} be the flip-flop hold time and T_{setup} be the flip-flop setup time. Two types of timing errors may exist:

(1) If $x_j + T_{hold} > x_i + \underline{d}(i,j)$, then when the positive clock edge arrives at FF_i, the data race ahead through the fast path, destroying the data at the input to FF_j before the clock gets there. When the clock edge finally arrives at FF_j, the wrong data are clocked through (Figure 4). Since the data are clocked through two FF's with one clock edge, this has been called **double-clocking**, or a **short path violation**.

(2) Analogously, **zero-clocking** or a **long path violation** can be used to designate the case when the data reach the FF_j too late relative to the next clock edge. This occurs when $x_i + \overline{d}(i,j) + T_{setup} > x_j + P$, where P is the clock period (Figure 5).

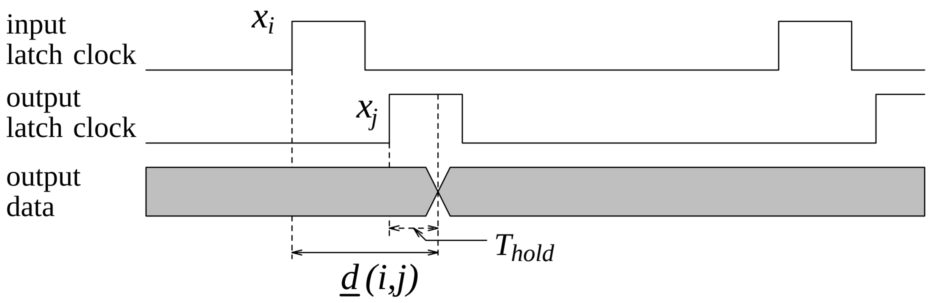

Figure 4: Double-clocking.

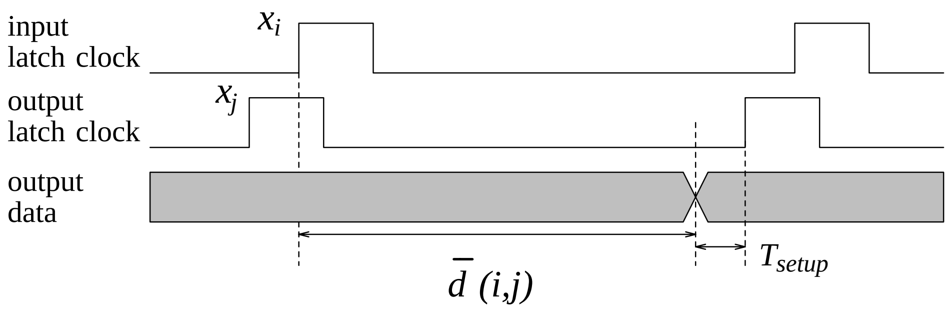

Figure 5: Zero-clocking.

2.2 The Clock Skew Optimization Problem

As can be seen from the previous section, to avoid long path and short path constraint violations, the following constraints apply:

$$\begin{aligned} x_i + \underline{d}(i,j) &\geq x_j + T_{hold}, \\ x_i + \overline{d}(i,j) + T_{setup} &\leq x_j + P \end{aligned} \tag{3}$$

The clock skew problem is solved under the assumption that the delays of the combinational segments are constant. The procedure for minimizing the clock period, P, involves the solution of the following optimization problem:

$$\begin{array}{lll} & \text{minimize} & P \\ \text{subject to} & x_i + \underline{d}(i,j) & \geq x_j + T_{hold} - \delta, \\ & x_i + \overline{d}(i,j) + T_{setup} & \leq x_j + P + \delta \end{array}$$

where the factor δ models the uncertainty in the skews. The explanation for including this factor is as follows: if one can guarantee that in the manufactured circuit, the skew at each FF k, $\hat{x}_k$, will be within the range $[x_k - \delta/2, x_k + \delta/2]$, where x_k is the design value of the skew, then the difference between any skews, $(\hat{x}_i - \hat{x}_j)$ in the manufactured circuit, must be within δ of the design value of $(x_i - x_j)$.

In the case where all gate delays are constant, the above optimization problem is a linear program in the skew variables and the clock period [9]. If we consider the gate delays as functions of the gate sizes, then $\underline{d}(i,j)$ and $\overline{d}(i,j)$ are functions of the gate sizes, and the optimization problem is considerably more complex. We will elaborate on this in Section 3.

2.3 Is Clock Skew Optimization Safe?

Several procedures for building clocking networks for nonzero skew are available in the literature [8, 12]. However, a common misconception that persists among circuit designers about changing clock skews is that it is believed to be an "unsafe" optimization, in that a small change in the gate/interconnect delays may cause a circuit with precariously small tolerances to malfunction. In fact, this is not so; one can build in safety margins [9, 10] that ensure that skewing errors do not disrupt circuit functionality. These safety margins ensure that the circuit will operate in the presence of unintentional process-dependent skew variations. Introducing deliberate delays within the clocking network has been a tactic that has long been used by designers [13], and this may be adapted to build fixed-skew clock networks.

As a proof of the practicality of this concept, a pipelined data buffer chip using the concept of skewed clocks was designed and fabricated in [14].

In fact, it is a misconception to believe that zero skew is entirely safe. To see this, consider a shift register consisting of register A whose output is connected to register B with *no* combinational logic between the two. Even for a circuit designed for zero skew, a small unintentional positive skew at register B will cause double-clocking, i.e., a short path constraint violation. Such problems may be avoided by the use of these safety margins and the introduction of deliberate nonzero skew: a small amount of deliberate positive skew at A provides an effective safety margin against short path violations.

3 Formulation of the Problem

As explained in Section 1.3, the use of clock skew optimization can help in achieving faster clock periods with lower area overheads. The combination of clock skew optimization and sizing into a single framework was thought to be a difficult (i.e., highly computational) problem since it was shown to be a signomial programming problem [9]. A recent approach in [15] used linear models to arrive at a solution, setting up the combined problem as a linear programming problem. However, the accuracy of piecewise linear models used in that work is limited, and hence it is desirable to investigate techniques that use the more accurate Elmore delay model [16] directly, as has been done for the sizing problem for combinational circuits in [3–6].

The combined problem of sizing and skew optimization is formulated as:

$$\begin{aligned}
\text{minimize} \quad & Area & & \\
\text{subject to } x_i + \overline{d}(i,j) + T_{setup} \quad & \leq & x_j + P + \delta & \qquad (4)\\
x_i + \underline{d}(i,j) \quad & \geq & x_j + T_{hold} + \delta & \qquad (5)\\
-X_{max} \quad & \leq & x_i \leq X_{max} & \qquad (6)
\end{aligned}$$

where $\overline{d}(i,j)$ and $\underline{d}(i,j)$ are functions of the gate sizes in the circuit, P is the specified clock period, and X_{max} is the maximum allowable skew magnitude. The *Area* objective function is approximated as the sum of all transistor sizes[1].

The area of the clocking network is not included here for two reasons. Firstly, it is difficult to derive a relation between the skews and the clocking network area (however, as shown in a

[1]For piecewise linear delay models, this problem is a linear program in the transistor size variables and the skew variables. It should be noted that most LP solvers require each variable to be strictly positive. In case some variables, such as the x_i's, may be negative, they can be represented as the difference between two positive variables [17].

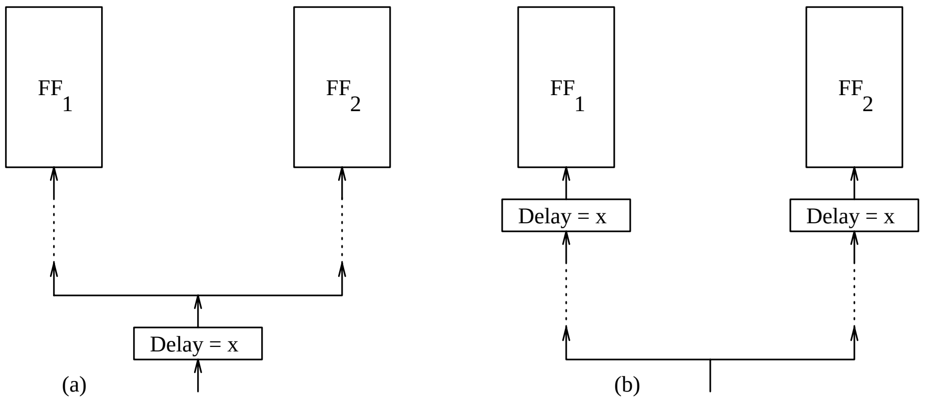

Figure 6: Clock network routing considerations.

following paragraph, we may add safeguards to prevent the clocking network area from becoming exceedingly large). Secondly, the complexity of routing the clock network is such that it is not possible to predict whether a nonzero skew clock tree will necessarily use up more routing resources than a zero-skew tree.

A simple model such as the sum of the skews would not work; consider, for example, the case where FF_1 and FF_2 are two flip-flops that are physically adjacent in the layout of the circuit and have the same skew. Then one may take advantage of their physical proximity by using the same set of buffers may be used to generate the skew for both, as shown in Figure 6(a), rather than wastefully adding extra buffers as shown in Figure 6(b).

We introduce the following mechanism to ensure that the expense of clock routing does not run amuck. We make the observation that for large clock skew magnitudes, the clock tree is likely to consume routing resources. Therefore, we introduce the constraint (6) to limit this expense. This constraint may easily be incorporated into our formulation.

Under the Elmore delay model, it can be shown [3] that the gate delays are posynomial functions[2] [18] of the gate sizes. A posynomial function in $\mathbf{y}$ can be transformed into a convex function in $\mathbf{z}$ using the mapping $y_i = e^{z_i}$. Based on this fact, it was pointed out in [9] that the above optimization problem is a *signomial* programming problem [18] and does not, in general, correspond to a convex program. This makes it difficult to arrive at a good solution to the problem.

If we examine each long path constraint, we note that it is of the form

$$\overline{d}(i,j) + x_i - x_j \leq K, \tag{7}$$

where $\overline{d}(i,j)$ is the maximum delay between flip-flops i and j (which is some posynomial function of the transistor sizes), x_i and x_j are clock delays to the source and destination flip-flops, and K is a constant. The left-hand side of this inequality is not a posynomial because of the negative coefficient of x_j. If the logarithmic substitution, $x = e^z$, were performed for each clock skew variable x and transistor size variable w in this inequality, the result would not be a convex constraint.

However, if we perform the substitution $w = e^z$ for each transistor width w appearing in $\overline{d}(i,j)$, while leaving x_i and x_j alone, then the result is a convex constraint: the left-hand side of

[2]A posynomial is a function g of a positive variable $\mathbf{w} \in \mathbf{R}^n$ that has the form $g(\mathbf{w}) = \sum_j \gamma_j \prod_{i=1}^{n} w_i^{\alpha_{ij}}$ where the exponents $\alpha_{ij} \in \mathbf{R}$ and the coefficients $\gamma_j > 0$. Roughly speaking, a posynomial is a function that is similar to a polynomial, except that (a) the coefficients γ_j must be positive. (b) an exponent α_{ij} could be any real number, and not necessarily a positive integer, unlike the case of polynomials.

the inequality is the sum of $\overline{d}(i,j)$, which is convex in the new z variables, and $x_i - x_j$, which is linear (hence convex), in the variables x_i and x_j.

We can generalize this concept to any optimization problem (not necessarily the problem that we are dealing with) that can be divided into two separate classes $w_1, \cdots, w_n$, and $x_1, \cdots, x_m$, with each constraint is of the form

$$P(w_1, ..., w_n) + C(x_1, ..., x_m) \leq K, \tag{8}$$

where P is a posynomial function, C is a convex function, and K is a constant. Such a problem can be transformed into a convex program by performing the substitution $w_i = e^{z_i}$ while leaving the x_i variables alone.

All of the above statements are valid for general sequential circuits, and not just pipelined circuits. Thus, we conclude that for general sequential circuits, the problem of adjusting both clock skews and transistor sizes to meet long path constraints while minimizing total gate area, corresponds to a convex optimization program.

Based on these observations, our approach to solving this problem is divided into two steps:

1. Neglect the short path constraints (which may be nonconvex [9]) and solve the combined sizing and skew optimization under long path constraints only. This is a unimodal problem, i.e., any local minimum is also a global minimum.

2. Resolve any short path constraints that are violated at the end of Step 1 by changing the topology of the circuit and adding buffers in an optimal manner using variation of techniques such as [11] (this is not addressed in our work).

Apart from easing the solution of the optimization problem, the above strategy has a useful side-effect. Since the presence of short path constraints results in a larger value of the optimal clock period over that achieved in the case where they are neglected, the method above will result in better clock periods than are achievable by keeping the circuit topology the same. It is also hoped that in practical circuits, the number of short path violations, and hence the number of buffers to be added, will be small.

The approach taken to solve Step 1 is an adaptation of the TILOS algorithm [3]. Although more exact methods such as those in [6] may be used to solve the convex programming problem exactly, these are not practical for use in this situation since the number of variables is too large for the exact algorithm to handle efficiently.

The optimization approach is divided into two stages that are repeated iteratively. In the first, violations of the clocking constraints must be detected. We present a modification of the PERT procedure for delay estimation, generalized to handle sequential circuits, in Section 4. Next, a critical path is defined and identified, and the size of the most sensitive gate on this path is bumped up by a small amount. The iterations continue until all long path-constraints are satisfied for the given clock period.

4 Detection of Long-Path Constraint Violations

4.1 The "Delay" of a Flip-Flop

PERT is a method for finding the longest/shortest path in a directed acyclic graph (DAG) that arises out of a system of difference constraints. It has been used extensively for delay estimation in combinational circuits. Here, we present a generalization that permits the use of the PERT method for timing analysis of acyclic sequential circuits.

The technique for representing combinational blocks by difference constraints, and therefore, by DAG's is well-known. For sequential circuits, one also has to deal with the problem of representing flip-flops. In this section, it is shown that the relationship between the input arrival time and the output arrival time (which we will refer to as the "delay") for flip-flops can also be represented by a difference constraint.

We point out here that the reference point for the arrival time in each combinational block is the zero skew. For example, an arrival time of a at the output of a flip-flop implies that the clock signal at the flip-flop has been skewed by a. If this flip-flop fans out to a single inverter with a unit delay, then the arrival time at the output of this inverter would be $a + 1$. Consequently, it can be seen that arrival times may be positive or negative. Note that this is a nonstandard definition of arrival time, but we use it since it measures the actual arrival time of a signal at any gate/flip-flop output in the circuit.

For acyclic pipelines, the composite set of these and the difference constraints for each gate can be represented by a DAG, on which PERT may be applied to solve the longest path problem.

Note that the word "delay," when used in reference to flip-flops, is applied in a loose sense here. The "delay" of a flip-flop as defined here is not the propagation delay of the gates within the flip-flop, but is a mathematical tool that can be used to apply PERT to a sequential circuit to check for delay violations in the presence of clock skews. It will be shown that one may assign a "delay" value to each flip-flop, and this value can be used in the same manner for the flip-flop as one would use the propagation delay for a gate during a PERT analysis.

It will now be shown that in an acyclic sequential circuit, a memory element is equivalent to a "delay" of $T_{setup} - P - \delta$, where T_{setup} is the setup time for the flip-flop and P is the applied clock period, and δ is the uncertainty in the clock skew. This arises as a direct result of applying clock skew optimization and the long path constraint given by (4).

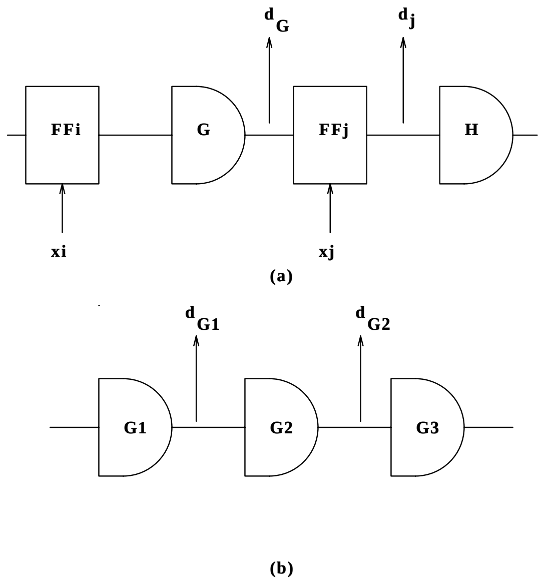

Figure 7: Modified PERT.

Consider Figure 7(a). We symbolically show the path from FF_i to FF_j as being represented by a single gate with delay $G_{maxdelay}$, which corresponds to the maximum combinational delay between these flip-flops. From (3) we have

$$x_i + G_{maxdelay} + T_{setup} \quad \leq \quad x_j + P + \delta \tag{9}$$

where x_i is the latest arrival time at the input of G, x_j is the latest arrival time at the input of gate H. Writing $x_i + G_{maxdelay}$ as d_G, the latest arrival time at the input of FF_j, we have the following difference constraint

$$\begin{aligned} d_G + T_{setup} &\leq x_j + P + \delta \\ i.e.\ x_j - d_G &\geq T_{setup} - P - \delta \end{aligned} \tag{10}$$

For a regular combinational gate, as shown in Figure 7(b), if d_{G1} and d_{G2} are the latest arrival times at the input of $G2$ and $G3$, then we have the difference constraint

$$d_{G2} - d_{G1} \quad \geq \quad delay(G2) \tag{11}$$

From the two difference constraints given above it can be seen that $G2$ and FF_j behave analogously, i.e., FF_j behaves like a gate with a delay of $T_{setup} - P - \delta$. PERT is accordingly carried out with gates and flip-flops being assigned delays as given by (10) and (11). It is noteworthy that the circuit will remain a DAG only when the pipeline is acyclic; if not, methods such as loop unrolling will have to be utilized.

Physically, the difference constraint for the FF represents a constraint that will lead to a calculation of the earliest time x_j (relative to the clock tick) at which a signal will arrive at the output of the FF in the presence of a skew.

As seen earlier, the value x_j also represents the required skew associated with the FF. Therefore, a violation is said to occur, if at the end of PERT, the arrival time at a primary output is greater than zero, or if the optimal skew at any flip-flop falls out of range of the constraint 6. Notice that if the arrival time at a primary output is less than or equal to zero, it corresponds to the existence of a nonnegative slack for the long path constraints; if the arrival time is positive, it constitutes a constraint violation.

At this point a backtrace is performed to find the *critical path*, i.e., starting from the primary output with the largest violation, a maximum delay path from a primary input is found. The most sensitive gate on the critical path is identified and is sized suitably in each iteration using a procedure described in Section 5. The details of the remaining steps are described in the following sections.

4.2 The Delay of a Gate

The method used here for modeling the delay of a gate has been used extensively and successfully in several gate sizing algorithms, and is not original. We present a description here for completeness. The delay of a gate can be characterized by its size and the capacitive load it drives. All transistors of the same type (n or p) within the gate are assumed to have the same size. The delay of each gate is calculated by replacing it by an equivalent inverter with parameters W_N and W_P corresponding to the equivalent n and p transistor sizes, respectively.

Each gate is replaced by an RC tree with a driving resistance R_{on} and a capacitive load C_{out}. The Elmore delay of the charging(discharging) network is taken to be the rise(fall) time of the gate. As has been done in previous work on sizing, the on-resistance, R_{on}, is inversely proportional to W_P (W_N) during the charging (discharging) transition. The capacitive load, C_{out}, is composed of two parts: C_{fanout}, the fanout capacitance, and C_{intr}, its intrinsic capacitance. The value of C_{fanout} of gate i is equal to its fanout gate capacitances and is given by

$$\begin{aligned} C_{fanout}(i) &= cap(i,1) + cap(i,2) + \cdots + cap(i,f) \\ &= \beta \cdot (W_{1,N} + W_{1,P}) + \beta \cdot (W_{2,N} + W_{2,P}) + \cdots + \beta \cdot (W_{f,N} + W_{f,P}) \end{aligned} \tag{12}$$

where $W_{1,N}, W_{1,P}, \cdots W_{f,N}, W_{f,P}$ are the sizes of the n and p transistors in the equivalent inverters corresponding to the gates that logic gate i fans out to, and β is a constant of proportionality.

The value of C_{intr} of gate i is proportional to its own size and is given by

$$C_{intr} = \alpha \cdot W_i \tag{13}$$

where W_i is the size of gate i, given by the sum of the n and p-transistor channel widths, and α is a constant of proportionality.

We can now write the delay of a gate by the following equation [3–6]:

$$Delay = R_{on} \cdot (C_{fanout} + C_{intr}) \tag{14}$$

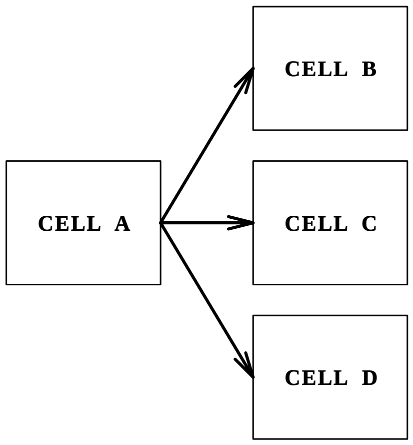

Figure 8: Example to show delay calculations.

For example, consider the gates in Figure 8. The rise delay of gate A is given by

$$\begin{aligned} \tau_r(A) &= R_{on}^A \cdot (C_{intr} + C_{fanout}) \\ &= \frac{1}{W_P^A} \cdot \left(\alpha W_A + C_{in}^B + C_{in}^C + C_{in}^D\right) \\ &= \frac{1}{W_P} \cdot (\alpha W_A + \beta W_B + \beta W_C + \beta W_D) \end{aligned} \tag{15}$$

where W_A, W_B, W_C, W_D are the gate sizes given by the sum of the n and p-channel transistor widths.

5 Sensitivity Computation

If the delay constraints cannot be satisfied by the current set of transistor sizes, the sizes of certain transistors in a gate are increased so as to effect the greatest decrease in delay with the smallest increase in area. This is called the *sensitivity* K of the gate. The sensitivity of a gate tells us how much its delay can be decreased if we increase its size by a unit amount. It is defined as $\frac{\Delta delay}{\Delta area}$, where $\Delta delay$ is the difference in the delay of the gate after the size has been bumped up, and $\Delta area$ is the increase in gate area. The optimization algorithm chooses the gate with the most negative sensitivity and bumps its size up by a fixed factor.

Since there are two delays associated with every gate, we have a *fall-sensitivity* K_n and a *rise-sensitivity* K_p which give the behavior of the delay when the n and p-channel widths are increased. They are given by the simple chain rule

$$\begin{aligned} K_n &= \frac{\delta delay}{\delta W_N} \cdot \frac{\delta W_N}{\delta area} \\ K_p &= \frac{\delta delay}{\delta W_P} \cdot \frac{\delta W_P}{\delta area} \end{aligned} \tag{16}$$

For a 2-input nand gate with all n-transistor sizes set to w_n and all p-transistor sizes set to w_p, $area = 2 \cdot w_n + 2 \cdot w_p$, $W_N = \frac{w_n}{2}$ and $W_P = w_p$ therefore $\frac{\delta W_N}{\delta area} = \frac{1}{4}$ and $\frac{\delta W_P}{\delta area} = \frac{1}{2}$.

To illustrate the calculation of sensitivity, consider the chain of inverters shown along with their output waveforms in Figure 9. In the succeeding discussion, the notation W_N^i and W_P^i will denote, respectively, the n and p transistor sizes of the i^{th} gate.

The rise delay at gate 3 is calculated as follows

$$\begin{aligned} \tau_{rise} &= R_{on}^3(P) \cdot C_{out}^3 + R_{on}^2(N) \cdot C_{out}^2 + R_{on}^1(P) \cdot C_{out}^1 + R_{in} \cdot C_{out}^{in} \\ &= \frac{1}{W_P^3} \cdot \left\{ \alpha \left(W_P^3 + W_N^3 \right) + C_{load} \right\} \\ &\quad + \frac{1}{W_N^2} \cdot \left\{ \alpha \left(W_P^2 + W_N^2 \right) + \beta \left(W_P^3 + W_N^3 \right) \right\} \\ &\quad + \frac{1}{W_P^1} \cdot \left\{ \alpha \left(W_P^1 + W_N^1 \right) + \beta \left(W_P^2 + W_N^2 \right) \right\} \\ &\quad + R_{in} \cdot \left\{ \beta \left(W_P^1 + W_N^1 \right) \right\} \end{aligned}$$

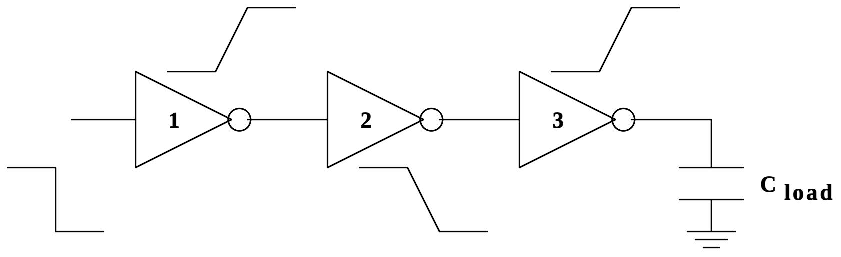

Figure 9: Chain of three inverters.

From this expression for delay the fall-sensitivity of gate 2, for example, is given by

$$K_n^2 = \frac{\beta}{W_P^1} - \frac{\alpha \cdot W_P^2 + \beta \cdot (W_P^3 + W_N^3)}{[W_N^2]^2}$$

Note that $\frac{\delta W_N^2}{\delta area} = 1$, since the primitive gate is itself an inverter. The general expression for sensitivity and expressions for $\frac{\delta W_N}{\delta area}$ and $\frac{\delta W_P}{\delta area}$ for various gates can similarly be evaluated.

6 Unifying Transistor Sizing and Clock Skew Optimization

It has already been seen that the use of transistor sizing with clock skew optimization is an effective method for reducing the clock period. In a multi-stage combinational circuit, if transistor sizing alone were applied, then each stage would be sized individually, and the circuit speed would be governed by the maximum delay of any combinational stage. By intentionally introducing clock skews it can be seen that for purposes of delay analysis, the multi-stage circuit may be viewed as a single block with flip-flops acting as modules with "delays." In the preceding sections, we have described individual pieces of the algorithm. This section presents an overview of the algorithm.

In the delay model presented in Section 4, each gate is represented by an equivalent inverter. The n-channel portion and the p-channel portion of each gate are sized independently of each other. Initially, all gate sizes are set to the minimum value. The sensitivity-based algorithm performs the following tasks in each iteration:

1. A timing analysis is carried out on the combinational circuit using PERT to identify paths that fail to satisfy the delay constraint. A violation is said to occur if the rise/fall delay at a PO is ≥ 0, or if the skew at a flip-flop exceeds the specified maximum skew. In each case, the delay of the path from the primary inputs to the flip-flop or PO must be reduced. Starting from the flip-flop or PO with the maximum violation, a critical path is traced back from the PO with the largest violation, to a PI. Since each gate is replaced by an equivalent inverter, the critical path is seen as an alternating sequence of rise and fall transitions.
2. The sensitivity of each gate along the critical path is computed, and the most sensitive gate is identified, so that its sizes may be bumped up. Depending on whether the gate to be sized is undergoing a rise (fall) transition, the p-channel (n-channel) width is bumped up.

Therefore, in each iteration, it is expected that the delay of the critical path will be reduced by a small amount. However, the change in the size of the most sensitive transistor for the current critical path is liable to increase the delays of other paths in the circuit, and hence, as in [3], it is important to use a small bumpsize.

The algorithm continues until all timing requirements are met, or until the sensitivity of the most sensitive gate in the most critical path is ≥ 0. At this stage, any further increase in channel widths results only in the increase in delay, and the specified clock period is unachievable. The process of calculating the optimal skews falls out as a natural consequence of this procedure: if the timing requirements are met, then the arrival time at the output of a flip-flop, as calculated by PERT, is the optimal skew to be applied to that flip-flop.

The pseudocode for the algorithm is shown below:

```
Read in circuit description;
Set all transistors to the minimum size;
While(TRUE) {
    Perform PERT.(Flip-Flops are seen as blocks with a delay of $T_{setup} - P - \delta$)
    If no timing violations exist
        exit.  /* timing and area specification achieved */
    Else {
        Trace the most critical path, $G_{PO_i} \cdots G_{PI_j}$
```

```
            Calculate rise and fall sensitivity of each gate in the path,
            If rise/fall sensitivity of most sensitive gate $\geq 0$,
               exit.  /* timing specification unachievable */
            Else, bump n/p-channel widths.
            If increase in area exceeds a limit, $Area_{spec}$,
               exit.  /* area specification unachievable */
        }
    }
```

Table 1: Input Circuit Description

	Number	Number	Number	Number	Number of
Circuit	of Gates	of Stages	of FF's	of PI's	PO's
add2†	15	1	5	3	2
inv_10†	10	1	2	1	1
r25†	30	1	10	5	5
r50†	55	1	10	5	5
r250†	265	1	30	15	15
r500†	520	1	40	20	20
mcnc	744	3	24	10	8
r_2	279	2	26	15	5
r_4	496	4	35	15	4
r_6	1370	6	78	25	10
r_10	1687	10	91	25	6
r_15	2986	15	142	25	9
r_20	4043	20	176	20	7

† single-stage pipelines

7 Experimental Results

The algorithm has been implemented in a C program, the SACS (Sizing And Clock Skew) optimizer. Experimental results are provided on several circuits, described in Table 1. For example, r_2 is a two-stage pipeline with a total of 279 gates, 26 flip-flops, 15 PI's and 5 PO's. The circuits marked with a "†" are single-stage pipelines. For these circuits, the results obtained with and without clock skew optimization must necessarily be identical since the clock skew adjustments can only be made on multi-stage pipelines.

Experimental results on these circuits, using transistor sizing with and without clock skew optimization, are presented in Table 2. The column labeled P_u corresponds to the clock period of an unsized circuit where all clock skews are set to 0, and P_{spec} is the specified clock period to be achieved. The corresponding percentage increases in circuit area as a result of sizing are, respectively, Δ_{area}^{nsk} and Δ_{area}^{sk}. The CPU times used by the program SACS for all of the circuits are also shown; these figures correspond to run-times on a DECstation 5000/133.

In Table 2, for both sizing with and without clock skew optimization, a bump size of 1.25 is used and no limit on the increase in area is imposed. (It may be recalled that the bump size is the factor by which the size of the most sensitive transistor is increased in each iteration.) It

Table 2: Combining Clock Skew Optimization and Transistor Sizing

Circuit	P_u in ns (Unsized Circuit Period)	P_{spec} in ns (Timing Spec.)	Δ_A^{nsk} (Area change w/ sizing only)	Δ_A^{sk} (Area change w/ sizing + skew)	T_{nsk} CPU Time (sizing only)	T_{sk} CPU Time (sizing + skew)
add2†	5.38	3.55	46.00%		0.36s	0.36s
inv_10†	5.76	4.25	55.13%		0.32s	0.32s
r25†	13.47	7.65	5.76%		0.35s	0.36s
r50†	25.12	13.05	18.34%		0.83s	0.83s
r250†	36.33	15.00	36.58%		19.37s	19.37s
r500†	48.59	30.00	3.79%		15.94s	15.94s
mcnc	77.74	18.0	- ‡	42.75%	-	107.6s
r_2	28.95	12.0	31.09%	23.69%	15.9s	14.8s
r_4	26.31	12.0	32.68%	23.98%	45.1s	55.9s
r_6	36.11	14.0	45.88%	32.10%	482.2s	396.7s
r_10	36.40	14.0	36.22%	26.51%	618.6s	488.6s
r_15	38.76	14.0	44.71%	31.82%	2885.5s	2305.5s
r_20	40.38	14.0	38.00%	26.51%	3656.9s	2830.8s

† single-stage pipelines ‡ specification could not be achieved

can be seen that in the case where clock skew optimization has been applied, the clock periods achieved are smaller than those achieved by not introducing clock skews. As expected, in the cases of the single-stage pipelines, clock skew optimization cannot be applied and the results for each case are the same. For the remaining circuits, it can be seen that wherever the specification, P_{spec}, is achieved by both the methods, $\Delta_A^{nsk} > \Delta_A^{sk}$. In other words, the increase in area is greater when clock skew optimization is not applied. It may also be observed that the period P_{spec} is sometimes unachievable using sizing alone, but is achieved by the use of skew optimization in addition to sizing.

For example, for r_20, a 20-stage pipeline with 4043 gates, the clock period of the unsized circuit is 40.38 ns. The specified clock period is 14 ns. The increase in area due to sizing without skew is 38%, as against about 26.5% for sizing+skew. The CPU times are seen to be similar in all cases. For some specifications for the circuits (as is shown later), the clock period specified here was not achievable by transistor sizing alone, demonstrating that by unifying transistor sizing and clock skew optimization, lower clock periods can be achieved.

Figures 10-12 indicate the improvement achieved when clock skew optimization is applied along with transistor sizing. Not only is the increase in area less, but the clock periods are also reduced further. For example, in Figure 12, even the best achievable clock period with sizing alone can be achieved by sizing+skew optimization at about half the cost, and that significantly lower clock periods are also achievable at a reasonable cost.

We caution the reader not to be misled by the fact that the two curves in each figure seem to be very close to each other. The curves show us that for a given timing specification, the area utilized by the sizing-only solution is, as expected, always larger than that for the sizing+skew solution. These differences are particularly acute when the circuit is designed for very tight timing constraints, where we try to push the limits of the achievable circuit speed.

In Figure 13, we display a plot showing the increase in area and CPU time versus the bump size for *r_2*. The bumpsize is varied between 1.01 and 15. As expected, a low bumpsize (close to,

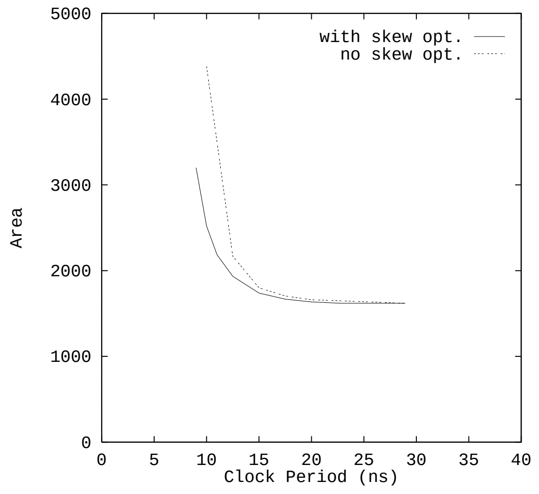

Figure 10: Increase in area vs. clock period for r_2.

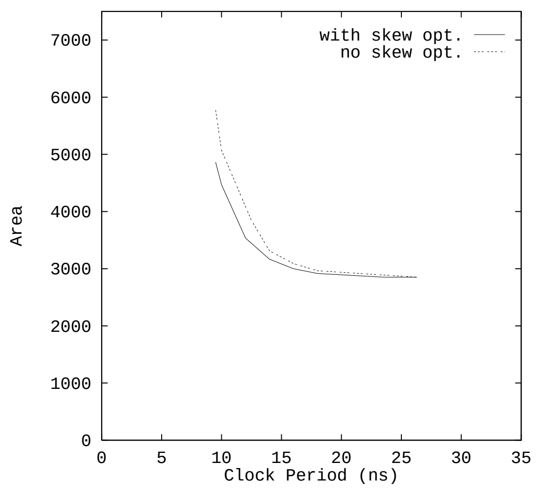

Figure 11: Increase in area vs. clock period for r_4.

but just larger than 1) provides the best quality solution, i.e., the solution with the least increase in circuit area. However, since the number of iterations for a very small bumpsize is liable to be large, the run-times increase greatly as the bumpsize is brought closer to 1. On the other hand, extraordinarily large values for the bump sizes lead to a larger increase in area. In fact, beyond a certain point, due to oversizing, high bumpsizes may actually cause the circuit delay to increase, rather than decrease (notice that large bumpsizes may even lead to a slightly larger execution time since a larger number of iterations may be required to achieve the specification). Moderately high bump sizes achieve the target much faster; however, the increase in area is not optimal. An optimal bump size must, therefore, be used. Experimentally, it has been found that using a bump size between 1.2 and 1.75 yields good results in terms of the trade-off between the computation time and the quality of the solution.

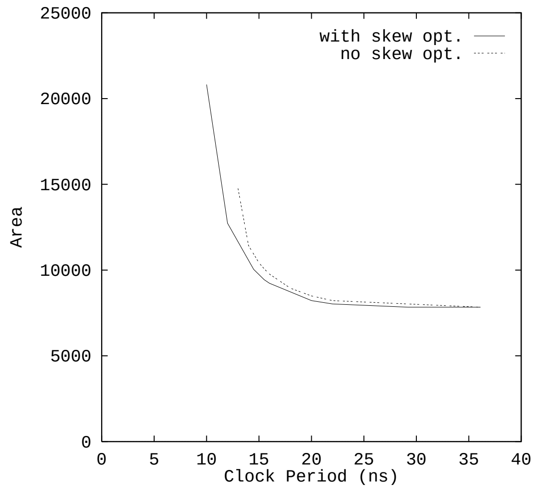

Figure 12: Increase in area vs. clock period for r_6.

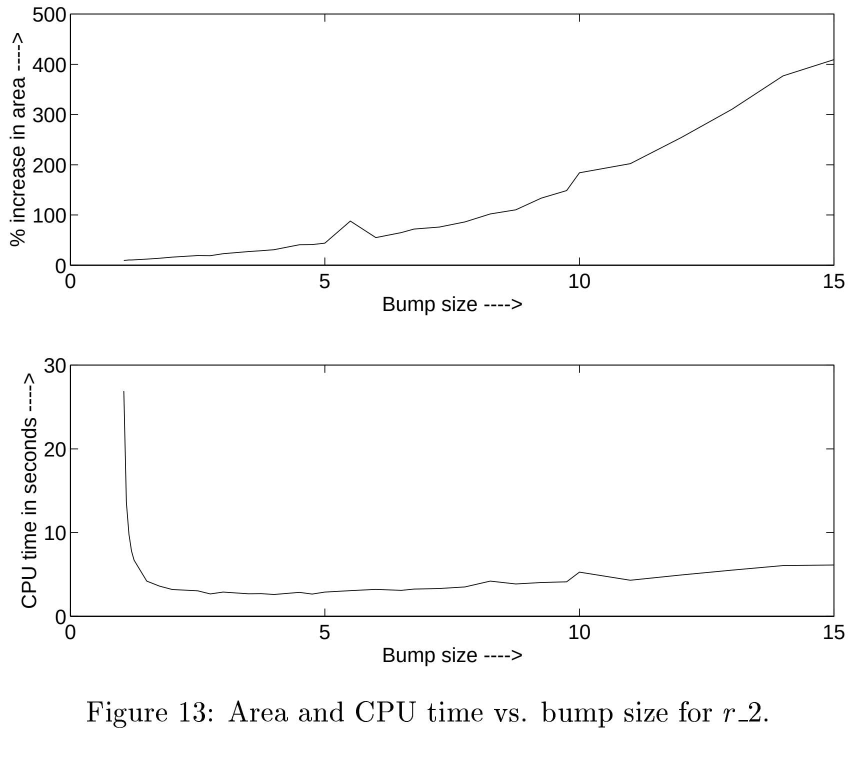

Figure 13: Area and CPU time vs. bump size for r_2.

The results in Table 2 correspond to unbounded values of X_{max} (defined in Equation (6)). The effect of X_{max} on the results for two circuits, r_4 and r_10, is shown in Table 3. The notation used here in each column is the same as that in Table 2. As expected, it is seen that as the value of X_{max} is decreased, the amount of sizing required to achieve a clock period increases.

This table also suggests a technique for controlling the expense of the clock network. The circuit could first be sized without bounding the value of X_{max}. The magnitude of the largest skew thus obtained is a rough indicator of how expensive the clock network would be. If this expense is too large, the value of X_{max} can be decreased and the optimization repeated to provide a tradeoff between the cost of building the clock network and that of sizing the circuit.

Table 3: Results of varying X_{max}

Circuit	P_{spec} in ns (Timing Spec.)	X_{max} (ns)	Δ_A^{sk} (Area change w/ sizing + skew)
r_4	22.5	∞	3.6
		3.0	4.5
Unsized		1.0	5.9
Area =		0.9	6.8
2851.2		0	6.9
	20.0	∞	26.6
		5.0	27.5
		1.0	36.0
		0.5	39.2
		0	49.3
	15.0	∞	222.0
		5.0	241.3
		1.0	274.3
		0.5	316.8
		0	359.0
	10.0	∞	1618.1
		5.0	1709.9
		1.0	1890.9
		0.5	1878.7
		0	2407.1
r_10	20.0	∞	346.7
		5.0	381.1
Unsized		1.0	622.0
Area =		0.5	774.1
9748.8		0	962.2
	15.0	∞	1755.1
		5.0	1864.4
		1.0	2465.3
		0.5	2803.9
		0	3404.5
	10.0	∞	11068.4
		5.0	11907.1
		1.0	14150.6
		0.5	18325.7
		0	unachievable

8 Conclusion

In this paper, a new approach to speeding up pipelined circuits using a conjunction of gate sizing and clock skew optimization has been presented. This problem was previously thought to be difficult under Elmore delays since it is a signomial programming problem [9]. We have shown that the sizing+skew problem under long path constraints only is equivalent to a convex optimization problem, and suggest that short path violations can be taken care of by considering each combinational block separately and adding buffers through variants of methods such as [11]. Note that this is a slight variation of the problem originally proposed in [9], since this involves the reconciliation of short path violations by changing the topology of the circuit by adding buffers. Experimental results validate the utility of combining the optimization methods of clock skew and sizing, and show that the conjunction of the two methods gives substantial improvements over using gate sizing alone.

In closing, we point to an interesting related issue, namely, the relationship between level-clocked circuits and the application of deliberate skew. A level-clocked circuit whose clock is high between $-T_{high}$ and 0 provides any skew between $-T_{high}$ and 0 at no cost. However, it does not allow the application of any skew outside that range. The work in [19] provides a technique for optimizing critical paths in level-clocked circuits.

9 Acknowledgments

The authors would like to acknowledge the anonymous reviewers for their helpful comments.

References

[1] S. Malik, K. J. Singh, R. K. Brayton, and A. Sangiovanni-Vincentelli, "Performance optimization of pipelined circuits," in *Proceedings of the IEEE International Conference on Computer-Aided Design*, pp. 410–413, 1990.

[2] N. V. Shenoy, R. K. Brayton, and A. L. Sangiovanni-Vincentelli, "Resynthesis of multi-phase pipelines," in *Proceedings of the ACM/IEEE Design Automation Conference*, pp. 490–496, 1993.

[3] J. Fishburn and A. Dunlop, "TILOS: A posynomial programming approach to transistor sizing," in *Proceedings of the IEEE International Conference on Computer-Aided Design*, pp. 326–328, 1985.

[4] D. Marple and A. E. Gamal, "Optimal selection of transistor sizes in digital VLSI circuits," in *Stanford Conference on VLSI*, pp. 151–172, 1987.

[5] J.-M. Shyu, A. L. Sangiovanni-Vincentelli, J. Fishburn, and A. Dunlop, "Optimization-based transistor sizing," *IEEE Journal of Solid-State Circuits*, vol. 23, pp. 400–409, Apr. 1988.

[6] S. S. Sapatnekar, V. B. Rao, P. M. Vaidya, and S. M. Kang, "An exact solution to the transistor sizing problem for CMOS circuits using convex optimization," *IEEE Transactions on Computer-Aided Design*, vol. 12, pp. 1621–1634, Nov. 1993.

[7] M. A. B. Jackson, A. Srinivasan, and E. S. Kuh, "Clock routing for high-performance IC's," in *Proceedings of the ACM/IEEE Design Automation Conference*, pp. 573–579, 1990.

[8] R.-S. Tsay, “An exact zero-skew clock routing algorithm,” *IEEE Transactions on Computer-Aided Design*, vol. 12, pp. 242–249, Feb. 1993.

[9] J. P. Fishburn, “Clock skew optimization,” *IEEE Transactions on Computers*, vol. 39, pp. 945–951, July 1990.

[10] R. B. Deokar and S. S. Sapatnekar, “A graph-theoretic approach to clock skew optimization,” in *Proceedings of the IEEE International Symposium on Circuits and Systems*, pp. 1.407–1.410, 1994.

[11] N. V. Shenoy, R. K. Brayton, and A. L. Sangiovanni-Vincentelli, “Minimum padding to satisfy short path constraints,” in *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design*, pp. 156–161, 1993.

[12] J. L. Neves and E. G. Friedman, “Circuit synthesis of clock distribution networks based on nonzero clock skew,” in *Proceedings of the IEEE International Symposium on Circuits and Systems*, pp. 4.175–4.178, 1994.

[13] K. D. Wagner, “Clock system design,” *IEEE Design and Test of Computers*, pp. 9–27, Oct. 1988.

[14] M. Heshami and B. A. Wooley, “A 250-MHz skewed-clock pipelined data buffer,” *IEEE Journal of Solid-State Circuits*, vol. 31, pp. 376–383, Mar. 1996.

[15] W. Chuang, S. S. Sapatnekar, and I. N. Hajj, “A unified algorithm for gate sizing and clock skew optimization to minimize sequential circuit area,” in *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design*, pp. 220–223, 1993.

[16] J. Rubinstein, P. Penfield, and M. A. Horowitz, “Signal delay in RC tree networks,” *IEEE Transactions on Computer-Aided Design*, vol. CAD-2, pp. 202–211, July 1983.

[17] P. E. Gill, W. Murray, and M. H. Wright, *Numerical Linear Algebra and Optimization*, vol. 1. Reading, Massachusetts: Addison-Wesley, 1991.

[18] J. Ecker, “Geometric programming: methods, computations and applications,” *SIAM Review*, vol. 22, pp. 338–362, July 1980.

[19] T. M. Burks, K. A. Sakallah, and T. N. Mudge, “Optimization of critical paths in circuits with level-sensitive latches,” in *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design*, pp. 468–473, 1994.