Utilizing the Retiming-Skew Equivalence in a Practical Algorithm for Retiming Large Circuits

Sachin S. Sapatnekar and Rahul B. Deokar

Department of Electrical and Computer Engineering

201 Coover Hall

Iowa State University

Ames, IA 50011.

1 Introduction

The importance of the issue of optimizing the timing behavior of VLSI circuits probably needs no introduction to any reader of this paper, and a great deal of effort has been invested into research in this field. This paper considers the method of retiming [1], which proceeds by relocating flip-flops within a network to achieve faster clocking speeds. A novel approach to retiming that utilizes the solution of the clock skew optimization problem [2] forms the backbone of this work.

The introduction of clock skew at a flip-flop has an effect that is similar to the movement of the flip-flop across combinational logic module boundaries. This was observed (but not proved) in [2], which stated that clock skew and retiming are continuous and discrete optimizations with the same effect. Although the designer can choose between the two transformations, these methods can, in general, complement each other. The equivalence between retiming and skew has been observed and used in earlier work, for example, in [3–5]. The contribution of this work is that it exploits this equivalence and presents a method that finds an optimal retiming efficiently, with a clock period that is guaranteed to be *at most* one gate delay larger than the optimal clock period. The method views the circuit hierarchically, first solving the clock skew problem at one level above the gate level, and then using local transformations at the gate level to perform retiming for the optimal clock period. The algorithm has been named ASTRA (A Skew-To-Retiming Algorithm).[1]

The clock skew problem is first solved using efficient graph-theoretic techniques in polynomial time. The idea of using graph algorithms is to take advantage of the structure of the problem to arrive at an efficient solution. Like [2], our technique is illustrated on single-phase clocked circuits containing edge-triggered flip-flops. The advantage of using this graph algorithm is that it not only minimizes the clock period, but that unlike a simplex-based linear programming approach, it also ensures that the difference between the maximum and minimum skews is minimized at the optimal clock period.

The complexity of solving the retiming problem at the gate level using the technique in [1] is $O(|G|^2 \log |G|)$, where $|G|$ is the number of gates in the circuit; this could be phenomenally large, and a "verbatim" implementation, such as that in SIS [6] is incapable of handling large circuits. However, a clever implementation, such as that proposed very recently in [7], can render the problem of finding a retiming of a circuit at the minimum clock period tractable. Another method proposed in [8] provides an efficient solution for the specific problem of circuits with unit-delay gates.

To place the our method in perspective with that in [7], it must be pointed out that the two were research efforts that were carried out independently and parallelly in time. While [7] performs the important task of debunking the myth that the Leiserson-Saxe algorithm cannot be applied to retime large gate-level circuits, our approach shows an alternative view to retiming, by way of clock skew optimization. Apart from taking different approaches to solving the retiming problem, the two also differ in that the work presented here also directly provides a solution to the problem of jointly finding an optimal retiming and optimal clock skews for a minimum period circuit. The work in [7] also performs minimum area retiming; however, we have currently left that as a topic for further research. The run-time of our approach and that in [7] are essentially the same for the minimum period retiming problem.

In this paper, the circuit is assumed to be composed of gates with constant delays. We assume the presence of an FF at each primary input and each primary output. The solution is divided

[1] "astra": (Sanskrit) a sophisticated weapon.

into two phases. In Phase A, the clock skew optimization problem is solved with the objective of minimizing the clock period, while ensuring that the difference between the maximum and the minimum skew is minimized. This difference provides a measure of how many gates have to be traversed in the next phase, and therefore, it is important that it be a small number. In Phase B, the skew solution is translated to retiming and some flip-flops are relocated across gates in an attempt to set the values of all skews to be as close to zero as possible. The designer may choose to achieve the optimal clock period by using a combination of clock skew and retiming; alternatively, any skews that could not be set exactly to zero could now be forced to zero. This could cause the clock period to increase; however, it is shown that this increase will be no greater than one gate delay.

The paper is organized as follows: the equivalence between retiming and clock skew is first shown in Sections 2 and 3. The algorithm for the clock skew optimization phase and a few related theoretical results are described in Section 4. Next, in Section 5, the process of transforming the solution of Phase A to a retiming solution is described, followed by a description of the properties of this solution in Section 6. Finally, we present experimental results in Section 7 and conclude the paper in Section 8.

2 Clock Skew Optimization and Retiming

In a sequential VLSI circuit, due to differences in interconnect delays on the clock distribution network, clock signals do not arrive at all of the flip-flops at the same time. Thus, there is a *skew* between the clock arrival times at different flip-flops. In a single-phase clocked circuit, in the case where there is no clock skew, the designer must ensure that each input-output path of a combinational circuit block has a delay that is less than the clock period. In the presence of skew, however, the relation grows more complex as one must compensate for this effect in designing the combinational circuits blocks.

One approach that has been followed by several researchers [9–13] is to design the clock distribution network so as to ensure zero clock skew. An alternative approach views clock skews as a manageable resource rather than a liability. It manipulates clock skews to advantage by intentionally introducing skews to improve the performance of the circuit. To illustrate how this may be done, consider the following example. In Figure 1, assume the delays of the inverters to be 1.0 unit each. The combinational circuit blocks CC_1 and CC_2 have delays of 3.0 and 1.0 units, respectively, and therefore, the fastest allowable clock has a period of 3.0 units. However, if a skew of +1.0 unit is applied to the clock line to flip-flop B, the circuit can run with a clock period of 2.0 units. This approach was formalized in the work by Fishburn [2]. The clock skew optimization problem was formulated as a linear program that may be solved to find the optimal clock period.

A second approach that is exploited here to improve the performance of the circuit is the procedure of retiming [1]. Retiming involves the relocation of flip-flops across logic gates to allow the circuit to be operated under a faster clock, without changing its functionality.

To illustrate how this may be done, consider the example in Figure 1, where it was seen that the clock period can be minimized to 2.0 units by introducing a skew of +1.0 unit at the flip-flop B. In an alternative approach, the period can still be minimized to 2.0 units by moving the flip-flop B to the left across the inverter I3. This results in the combinational circuit blocks CC_1 and CC_2 having delays of 2.0 units each as seen in Figure 2. This approach is formalized in [1].

If one were to imagine the circuit as being drawn with its inputs to the left and outputs to the right, then the conversion of a negative (positive) skew to zero skew would involve the relocation of flip-flops to the right (left). In this paper, we will use the terms "right" and "left" to denote the direction of signal propagation and the direction opposite to that of signal propagation, respectively.

3 Equivalence between clock skew and retiming

A more formal presentation of the equivalence between clock skew and retiming is presented here. Consider a flip-flop j in a circuit, as shown in Figure 3(a). For every combinational path from a flip-flop i to j with delay $d(i, j)$, the following constraints must hold to prevent zero-clocking and double-clocking, respectively.

$$
\begin{aligned}
x_i + d(i,j) + T_{setup} &\leq x_j + P \\
x_i + d(i,j) &\geq x_j + T_{hold}
\end{aligned}
\tag{1}
$$

where x_i and x_j are the skews at flip-flops i and j, respectively. Similar constraints can be written for every combinational path from flip-flop j to k with delay $d(j,k)$.

Theorem 1 For a circuit that operates at a clock period P, satisfying the double-clocking and zero-clocking delay constraints,

(a) retiming a flip-flop by moving it to the left across a gate of delay d_1 is equivalent to decreasing its skew by d_1.

(b) retiming a flip-flop by moving it to the right across a gate of delay d_2 is equivalent to increasing its skew by d_2.

Proof:

We will now prove statement (a) with the help of Figure 3(b); the proof to (b) is analogous, and can be seen via Figure 3(c). Consider the case where flip-flop j is moved to the left across gate G_l that has delay d_1. For a combinational path from a flip-flop i to j, if x_j and x'_j are, respectively, the skews at flip-flop j before and after the relocation operation, then the following relationships must be satisfied:

$$
\begin{aligned}
x_i + [d(i,j) - d_1] + T_{setup} &\leq x'_j + P \\
x_i + [d(i,j) - d_1] &\geq x'_j + T_{hold}
\end{aligned}
$$

From Equation (1), this implies that setting $x'_j = x_j - d_1$ achieves the same effect. Similarly, for a combinational path from j to a flip-flop k, we have

$$
\begin{aligned}
x'_j + [d(j,k) + d_1] + T_{setup} &\leq x_k + P \\
x'_j + [d(j,k) + d_1] &\geq x_k + T_{hold},
\end{aligned}
$$

which implies that setting $x'_j = x_j - d_1$ in the original circuit achieves the same effect. □

Therefore, if one were to calculate the optimal clock skews, one could retime the circuit by moving flip-flops with positive(negative) skews to the left(right) until the skews at the flip-flops are nearly equal to zero. It must be noted that since gate delays take on discrete values, it is not possible to guarantee that the skew at a flip-flop can always be reduced to zero through retiming operations.

An alternative view of the same procedure is as follows. Retiming may be thought of as a sequence of movements of flip-flops across gates (Theorem 9.1 in [14]). We may start from the final retimed circuit, where all of the skews are zero, and the zero-clocking constraints are met, and perform the sequence of movements in reverse order. This procedure can be used to move all flip-flops back to their initial locations, using Theorem 1 to keep track of the changed clock skews at each flip-flop. Therefore, the optimal retiming is equivalent to applying skews at the inputs of flip-flops.

Note that the optimal clock period provided by the clock skew optimization procedure must, by definition, be no greater than the clock period for the set of clock skews thus obtained. Any differences arise due to the fact that clock skew optimization is a continuous optimization, while retiming is a discrete optimization. The following corollary follows:

Corollary 2 : The clock period obtained by an optimal retiming can be achieved via clock skew optimization. The clock period provided by the clock skew optimization procedure is less than or equal to that provided by the method of retiming.

4 Phase A: Optimizing clock skews

Given a combinational circuit segment that lies between two flip-flops i and j, as shown in Figure 4, if x_i and x_j are the skews at the two flip-flops, then the following inequations must be satisfied:

$$\begin{aligned} x_i + \underline{d}(i,j) &\geq x_j + T_{hold}, \\ x_i + \overline{d}(i,j) + T_{setup} &\leq x_j + P \end{aligned} \qquad (2)$$

where $\underline{d}(i,j)$ ($\overline{d}(i,j)$) is the minimum (maximum) delay of any combinational path between flip-flops i and j. In [2], the clock skew problem for minimizing the clock period is solved by solving the following linear program:

$$\begin{aligned}
\text{minimize} \quad & P \\
\text{subject to} \quad x_i - x_j & \geq T_{hold} - \underline{d}(i,j) \\
x_j - x_i + P & \geq T_{setup} + \overline{d}(i,j)
\end{aligned} \tag{3}$$

for every pair, (i,j) of flip-flops such that there is at least one combinational path from flip-flop i to flip-flop j.

In this work, we will consider single-sided constraints only, and will ignore the short path constraints. We thus obtain the clock skews that correspond to the minimum clock period. Our strategy is to transform the skew solution to a retiming solution to achieve the minimum clock period. The rationale behind this approach is that, as will be shown in the next section, the clock period obtained thus is smaller than that obtained with the inclusion of double-sided constraints. This minimum clock period can be preserved while reconciling short path and logic signal separation constraint violations [15] by using an algorithm for minimum padding, such as the one in [16].

4.1 Solution to the clock period optimization problem

4.1.1 Formation of the constraint graph

The linear program (3) without the short path constraints is rewritten as

$$\begin{aligned}
\text{minimize} \quad & P \\
\text{subject to} \quad x_j - x_i + P & \geq T_{setup} + \overline{d}(i,j)
\end{aligned} \tag{4}$$

Notice that for a constant value of P, the constraint matrix reduces to a system of difference constraints which can be represented by a constraint graph [17]. A feasible solution to the linear program exists if the corresponding constraint graph $G_1(P)$ contains no positive cycles, and the minimum clock period corresponds to the smallest value of P at which no positive cycle exists.

The skews at all primary inputs and primary outputs are assumed to be zero; this is represented by a host node in the constraint graph, similar in principle to the notion in [1].

Observe that the constraint set of the linear program (4) is a subset of the constraint set of the linear program (3). Therefore, the optimal period for the LP above must be less than or equal to that for the LP that handles double-sided constraints.

If $\overline{d}(i,j)$ is finite, then a directed edge between x_i and x_j are constructed in $G_1(P)$ in accordance with the long path delay constraint.

4.2 Calculating the worst-case flip-flop-to-flip-flop delays

For any input i, the procedure [2] for computing $\overline{d}(i,j)$ for all j involves setting the arrival time at input i to zero, and that at all other inputs to $-\infty$. The resulting signal arrival time at each output j, found using PERT [18], is the value of $\overline{d}(i,j)$. However, if this procedure were to be performed directly, it would lead to large computation times.

It was observed during the symbolic propagation of constraints in [19] that in most cases, a flip-flop at the input to a combinational block exercises only a small fraction of all of the paths between the inputs of the combinational block to the outputs. Based on this observation, we develop an efficient procedure for calculating the values of $\overline{d}(i,j)$. It was found that the use of this procedure gave run-time improvements of several orders of magnitudes over the direct multiple applications of PERT described in the previous paragraph.

Since we will be dealing with combinational blocks only here, in this subsection *only*, we will refer to the flip-flops at the inputs(outputs) of a combinational block as primary inputs(outputs).

The level number, level(k), of each gate k in the circuit is first computed by a single PERT run; the level number is defined as the largest number of gates from a primary input to the gate, inclusive of the gate. In other words, the level number of a gate is found by a topological ordering algorithm. To find $\overline{d}(i,o)$, the largest delay from primary input i to all primary outputs o, we conduct an event-driven PERT-like exercise starting at flip-flop i, as described in the following piece of pseudocode. During the process, we maintain a set of queues, known as level queues. The level queue indexed by k contains all gates at level k that have had an input processed.

```
Initialize all level queues to be empty;
currentlevel = 0;
currentgate = flip-flop i;
while (currentgate != nil) {
    if (currentgate ≠ i)
        for (all fanins j of currentgate)
            d̄(i,currentgate) = max(d̄(i, j)) + delay(currentgate);
    else
            d̄(i,i) = 0;
    for (all fanouts k of currentgate)
        Append gate k to the level queue indexed by level(k), if it
            does not already lie on the queue;
    if (all level queues are not empty) {
        currentlevel = lowest level number whose level queue is nonempty;
        currentgate = head of the level queue for currentlevel;
        delete currentgate from its level queue;
    }
    else
        currentgate = nil;
}
```

At each step, an element from the lowest unprocessed level is plucked from its level queue, and the worst-case delay from flip-flop i to its output is computed. All of its fanouts are then placed on their corresponding level queues, unless they have already been placed on these queues. Note that by construction, no gate is processed until the delay to all of its inputs that are affected by flip-flop i have been computed, since such inputs must necessarily have a lower level number.

4.3 Theoretical results on the optimal clock period

The following theoretical results are proved for the optimal clock period:

Lemma 3 : If $P = P_1$ does not permit a feasible solution to the linear program (4), then nor does $P = P_2 < P_1$.

Proof : If $P = P_1$ is such that it does not permit a feasible solution to the linear program, then the constraint graph $G_1(P_1)$ has a positive cycle, C.

If the cycle C has r edges then

$$\sum_{q\in C} w_q(P_1) \geq 0$$

$$\sum_{q\in C}(T_{setup} + \overline{d}_{ij} - P_1) \geq 0 \tag{5}$$

where $q = (i,j)$ is an edge in $G_1(P)$.

If $P_2 < P_1$, then since the weight of the cycle C is

$$\sum_{q\in C}(T_{setup} + \overline{d}_{i,j} - P_2) > \sum_{q\in C}(T_{setup} + \overline{d}_{i,j} - P_1) \geq 0, \tag{6}$$

C will be a positive cycle in $G_1(P_2)$ too. Hence, P_2 does not permit a feasible solution to the linear program (4). $\square$

Corollary 4 : If $P = P_1$ permits a feasible solution to the linear program (4), then so does $P = P_2 > P_1$.

Proof : Assume that P_2 does not permit a feasible solution. Therefore, by Lemma 3, nor does P_1, which is a contradiction. $\square$

Lemma 5a : An upper bound, P_{high} on the clock period, P_{opt}, is given by

$$P_{opt} \leq P_{high} = \max_{q=(i,j)\in G_1(P)}(\overline{d}(i,j) + T_{setup}) < \infty \tag{7}$$

where $q = (i,j)$ is an edge in $G_1(P)$.

Proof : Suppose all clock skews are set to 0. Then from the equation for zero-clocking constraints, we have

$$P \geq (\overline{d}(i,j) + T_{setup}) \;\; \forall (i,j) \in G_1(P) \tag{8}$$

Therefore,

$$P = \max_{q=(i,j)\in G_1(P)}(\overline{d}(i,j) + T_{setup}) < \infty \;\; \forall (i,j) \in G_1(P) \tag{9}$$

is a valid solution, and is thus an upper bound on the optimal clock period. $\square$.

Lemma 5b : A lower bound, P_{low} on the optimal clock period, P_{opt}, is given by

$$P_{opt} \geq P_{low} = \max \left[\min_{q=(i,j) \in G_1(P)} (\overline{d}(i,j) + T_{setup}), \overline{d}_{\text{loop}} \right] \geq 0 \tag{10}$$

where $q = (i,j)$ is an edge in $G_1(P)$, and $\overline{d}_{\text{loop}}$ is defined as the largest weight of any edge in $G_1(P)$ from any node (including the host node) to itself, if such a loop exists, and 0 otherwise.

Proof : The weight of any edge that starts and ends at the same node, $\overline{d}(i,i) + T_{setup}$, is clearly a lower bound on the clock period. If no such edge exists, then the lower bound is calculated as follows.

Let C be the critical cycle at the optimal clock period, P_{opt}, i.e., a cycle with weight zero. (Clearly one such cycle must exist, or else P_{opt} could be reduced further.) Let k be the number of edges in C. Therefore

$$\begin{aligned}
0 &= \sum_{(i,j) \in C} \left[(\overline{d}(i,j) + T_{setup} - P_{opt}) \right] \\
\text{i.e., } P_{opt} &= \frac{1}{k} \cdot \sum_{(i,j) \in C} \left[\overline{d}(i,j) \right] + T_{setup} \\
\text{i.e., } P_{opt} &\geq \min_{(i,j) \in C} \overline{d}(i,j) + T_{setup} \\
\text{i.e., } P_{opt} &\geq \min_{(i,j) \in G_1(P)} \overline{d}(i,j) + T_{setup}.
\end{aligned}$$

The nonnegativity of the right hand side is trivial. □

Theorem 6 : A finite solution to the clock period minimization problem exists.

Proof : From Lemma 5, the solution P to the clock period minimization problem is bounded by $0 \leq P_{low} \leq P \leq P_{high} < \infty$. □

4.4 The clock skew optimization procedure

The skeletal pseudocode describing the algorithm for finding the optimal clock period proceeds as follows, using a binary search on the value of the clock period.

Construct the constraint graph;
$P_{max} = P_{high}$; ...(Lemma 5a)
$P_{min} = P_{low}$; ...(Lemma 5b)
while $(P_{max} - P_{min}) > \epsilon$ {
    $P = (P_{max} + P_{min})/2$;
    if $G_1(P)$ has a positive cycle
        $P_{min} = P$;
    else
        $P_{max} = P$;
}

In the above algorithm, the presence of a positive cycle in $G_1(P)$ may be tested using the Bellman-Ford algorithm [17]. If the skews are initialized to 0, the Bellman-Ford solution achieves the objective of minimizing $|x_{i,max} - x_{i,min}|$ [17]. On a graph with V vertices and E edges, the computational complexity of this algorithm is $O(V \cdot E)$. The number of iterations is $(P_{high} - P_{low})/\epsilon$.

The time required to form the constraint graph may be as large as $|f| \cdot |G|$, where $|f|$ is the maximum number of inputs to any combinational stage, though in practice, it is seen that this upper bound is seldom achieved.

Therefore, the iterative procedure above, when carried to convergence, provides the solution to the linear program (4) with a worst-case time complexity of

$$O\left(F \cdot E \cdot log_2\left[(P_{high} - P_{low})/\epsilon\right] + |f| \cdot |G|\right) \tag{11}$$

where F is the number of flip-flops in the circuit, E is the number of pairs of flip-flops connected by a combinational path, and P_{high} and P_{low} are as defined in Lemma 5, and ϵ, defined in the pseudocode above, corresponds to the degree of accuracy required.

We point out that for real circuits, $E = O(F)$.

We caution the reader that the complexity shown above is not a genuine indication of the complexity if the implementation is cleverly carried out, using back-pointers during the Bellman-Ford process [20] and the procedure in Section 4.2 for $\overline{d}(i,j)$ calculations.

In the solution found above, all skews must necessarily be positive, since the weights of each node in the Bellman-Ford algorithm was initialized to zero. Also, in general, the skew at the host node (corresponding to primary inputs and outputs) could be nonzero. Our objective is to ensure a zero skew at the primary input and output nodes since we do not have the flexibility of retiming these, and hence we modify this solution. Note that if $[x_1, \cdots x_n]$ is a solution to a system of difference constraints in $\mathbf{x}$, then so is $\mathbf{x}' = [(x_1+k), (x_2+k), \cdots, (x_n+k)]$. Therefore, by selecting k to be the negative of the skew at the host node, a solution $\mathbf{x}'$ with a zero skew at the host is found.

5 Phase B: Skew minimization by retiming

5.1 Introduction

In Phase B, the magnitudes of the clock skews obtained from Phase A are reduced to zero by applying retiming transformations. This employs relocation of the flip-flops across logic gates while maintaining the optimal clock period previously found. After the skew magnitudes have been reduced by as much as possible, the retimed circuit may be implemented by applying the requisite skews at a flip-flop (to get the minimum achievable clock period) or by setting all skews to zero (to get a clock period that is, as will be shown in Section 6, no more than one gate delay above the optimum).

Since any flip-flop to be moved must have a nonzero skew, we divide the relocations into one of the two following categories:

- Flip-flops with negative skews

- Flip-flops with positive skews

Before we proceed, we will state the following result, which is something of a generalization of Theorem 1. The theorem pertains to Figure 5, where the flip-flops at the input are retimed across a combinational block to its outputs.

Theorem 7:

(a) Retiming transformations may be used to move flip-flops from all of the inputs of any combinational block to all of its outputs. The equivalent skew of the relocated flip-flop at output j, *considering long path constraints only*, is given by

$$x_j = \max_{1 \le i \le n} (x_i + \overline{d}(i, j)) \tag{12}$$

where the $x_i, 1 \le i \le n$ are the skews at the input flip-flops, and x_j is the equivalent skew at output j, and $\overline{d}(i, j)$ is the worst-case delay of any path from i to j.

(b) Similarly, flip-flops may be moved from all of the outputs of any combinational block to all of its inputs, and the equivalent skew at input k, considering long path constraints only, is given by

$$x_k = \min_{1 \le j \le m} (x_j - \overline{d}(k, j)) \tag{13}$$

where the $x_j, 1 \le j \le m$ are the skews at the input flip-flops, and x_k is the equivalent skew at input k, and $\overline{d}(k, j)$ is the worst-case delay of any path from k to j.

The proof is omitted and proceeds along the lines of that for Theorem 1. We point out here that in general, it is not possible to come up with an equivalent skew value that satisfies both long and short path constraints. For example, when we consider short path constraints, moving flip-flops from the input to the output requires that the new skew be

$$\min_{1 \le i \le n} (x_i + \underline{d}(i, j))$$

(using the same notation as Theorem 7(a)), which is incompatible with the requirement stated above except in the special case where all paths from i to j have the same delay. However, this is not a serious problem for us here since we are only considering the long-path constraints here, as stated in Section 4.

While the preceding paragraph may superficially seem to contradict Theorem 1, which is valid for both long path and short path constraints, we reassure the reader that this is not so. Theorem 1 can be considered to be a special case where the combinational subcircuit consists of only one gate, where clearly the longest and shortest path delays of the combinational subcircuit are equal.

5.1.1 Case 1: Negative skew reduction

Consider the case of a flip-flop j shown in the Figure 6(a) that has a negative skew at the conclusion of Phase A.[2] If we consider the gate p to which j fans out, we may find its transitive fanins and identify the flip-flops at the input of the combinational subcircuit to which p belongs. Through retiming operations, it is possible to transform the circuit in Figure 6(a) to the one in Figure 6(b); the equivalent skews at each flip-flop in Figure 6(b) are calculated. At this point, it need only be noted that the equivalent skews for these flip-flops are found without physically moving them to the gate inputs.

To see why the transformation from Figure 6(a) to Figure 6(b) is always possible, note that the combinational block shown in the figure can be replaced by a set of "input-output" delay constraints. We may then apply the result of Theorem 7(a) to obtain the equivalent skews; the complete procedure will be described later.

There now exists a set of n "virtual" flip-flops[3] at the input to gate p, as shown in Figure 6(b). Let $x_1, x_2, \cdots, x_n$ be the skews of these flip-flops. They must satisfy the constraints:

$$x_k + delay \quad \leq \quad x_i + P \; \forall \; 1 \leq k \leq n \tag{14}$$

where P is the clock period, x_i the skew at an output flip-flop, FF_i (not shown), of the combinational

[2] We assume here that each flip-flop fans out to exactly one gate. If the fanout of a flip-flop is larger than one, then it is replicated at each fanout branch. The replicated flip-flops have exactly one fanout gate, and each such flip-flop is considered in turn.

[3] We refer to these flip-flops as "virtual" flip-flops because we do not physically move them to the input of gate p at this point.

block to which $FF_1 \cdots FF_n$ are input flip-flops, and $delay$ is the largest combinational delay from the input of gate p to FF_i.

Obviously, from the above constraints:

$$\max_{1 \leq k \leq n} (x_k) + delay \leq x_i + P \tag{15}$$

Now, there can exist two scenarios:

(1) All of the n flip-flops at the inputs have negative skews. In this case, the maximum of all the negative skew flip-flops is negative and hence the set of flip-flops may be moved across the gate p, as shown in Figure 6(c). If the sign of the skew were to change after the relocation, the relocation would not be carried out unless it reduced the magnitude of the skew, and if not, it would be left in its current location with a skew of $\max_{1 \leq i \leq n}(x_i)$. For example, a flip-flop with a skew of -0.75, to be moved across a gate with unit delay, would have a new skew value of 0.25; such a relocation would be desirable since it reduces the magnitude of the clock skew (this idea is better understood in the context of Lemma 8 and Theorem 9, to be stated and proved later in this paper). However, it would be undesirable to move a flip-flop with skew -0.25 across a unit delay gate. Therefore, in either case, this effects a reduction in the magnitudes of the skew values, as is desirable.

(2) One or more of the "virtual" flip-flops has a positive effective skew. In this case, the skew at the flip-flop j under consideration may be set to zero without violating any timing constraints, since the maximum skew at the input to gate p would be unchanged by this operation. One example where this may occur is when one of the inputs to the combinational block is a primary input of the sequential circuit; the equivalent skew of the corresponding virtual FF at the gate input would be positive. Note that due to this, it is never necessary for a primary input to be relocated, which is in conformance with our assumption that these are immovable.

5.1.2 Case 2: Positive skew reduction

In the case of a flip-flop j that has a positive skew at the end of Phase A, as shown in Figure 7(a),[4] the procedure parallels that described above. Through retiming operations, it is possible to transform the circuit in Figure 7(a) to the one in Figure 7(b); the equivalent skews at each flip-flop in Figure 7(b) are calculated using Theorem 7(b); the precise procedure will be described later. Therefore, at the output of the gate p, there now exists a set of n "virtual" flip-flops as shown in Figure 7(b), with effective skews $x_1, x_2 \ldots x_n$. The skews at these flip-flops need to satisfy the constraints:

$$x_i + delay \quad \leq \quad x_k + P \; \forall \; 1 \leq k \leq n \tag{16}$$

where x_i the skew at an input flip-flop, FF_i (not shown), of the combinational block to which $FF_1 \cdots FF_n$ are output flip-flops, and $delay$ is the largest combinational delay from FF_i to the output of gate p.

The above constraints reduce to

$$x_i + delay \leq \min_{1 \leq k \leq n} (x_k) + P \tag{17}$$

As before, we may have one of two scenarios:

(1) If all of the n flip-flops in the array have positive skews, the minimum of all the positive skew flip-flops is positive and hence the set of flip-flops may be moved across the gate p, as illustrated in Figure 7(c). If the sign of the skew were to change after the relocation, the relocation would not be carried out unless it reduced the magnitude of the skew. Therefore, in either case, this effects a reduction in the positive skew values, as is desirable.

One may also take advantage of slacks in the combinational paths to reduce the skews at flip-flops. If input r to gate p has a slack, $slack(r)$ (i.e., the worst-case delay at input r could

[4] Each flip-flop has exactly one fanin, and hence the problem of multiplicity described in a footnote for Case 1 is irrelevant.

have been increased by $slack(r)$ without changing the worst-case delay to the output of gate p), then the skew may be further reduced by $slack(r)$. If $slack(r) > min(x_i) - d$, the skew is set to zero; if not, it is set to $min(x_i) - d - slack(r)$.

(2) If one or more of the "virtual" flip-flops has a negative skew value, then the skew at the flip-flop j under consideration is set to zero. This violates no timing constraints, since it leaves the minimum skew at an output of gate p unchanged.

5.2 Minimization procedure for case 1

The steps involved in minimizing the skew magnitudes for flip-flops with negative skews are outlined below:

Step 1 All flip-flops in the circuit with negative skews are placed on a queue, Q. For each flip-flop, we consider one fanout at a time; in other words, the situation shown in Figure 8(a) is transformed to the form in Figure 8(b).

Step 2 Let j be the flip-flop that is currently at the head of the queue, and p the gate that it fans out to. We will assume for now that p is a gate; the case where it is the input of a flip-flop will be dealt with separately. The equivalent skew at every other fanin node of p is found. We do not physically move flip-flops to the inputs of gate p at this time, and are hence we imagine that the inputs to gate p are a set of "virtual" flip-flops.

The equivalent skews are found as follows. Consider a flip-flop j with negative skew, as shown in Figure 9. The gate p to which it fans out to is added to the tail of a queue R.[5] A reverse PERT is employed to backtrace along the fanin cone of gate p. When a gate is encountered, it is added to the queue. In Figure 9, gate z is first added to R, and in the next step, gate y is added. The backtrace continues until a flip-flop is encountered. In the example in Figure 9, the backtrace terminates when flip-flop x is encountered. During this process, we keep track of the worst-case delay, d, to gate p. As a consequence of Theorem 3, if the skew (calculated

[5]Note that the queue R is distinct from the queue Q.

in Phase A) at flip-flop x is t units, then its equivalent skew at the input to gate p is $t + d$ units.

Step 3 If any equivalent skew at a "virtual" flip-flop is positive, then the skew at j is set to zero and it is not relocated; if not, the skew after retiming is found using the criteria described earlier. If the magnitude of this skew is smaller than the current skew at j, then j and all of the "virtual" flip-flops at the input to p are retimed across p. (Notice that if the skew changes sign after retiming, then the magnitude of the retimed skew could become larger. Only those sign-changing moves that reduce the skew magnitude are permitted.) Note that the motion of the "virtual" flip-flops to their new location may entail replicating these flip-flops, as shown in Figure 8. For example, if flip-flop j were to be moved across gate p, a new flip-flop would have to be created at y_2 with an equivalent skew corresponding to the skew of flip-flop x, retimed to position y_2. The new skews are found as explained in the previous section. Any such flip-flops that have a negative skew (as will happen most of the time, unless relocation changed the sign of the skew) are now placed at the tail of the queue, Q, and are processed later.

Step 4 If the retimed flip-flop has a negative skew, it is placed at the tail of Q.

Step 5 If Q is not empty, go to Step 1; if not, the magnitudes of all negative skew values have been minimized.

In Step 2 above, if the flip-flop j fans out to another flip-flop, which we shall call k, then since there is no combinational delay between the flip-flops, and retiming preserves the zero-clocking constraints, it must be true that

$$\begin{aligned} x_j + T_{setup} &\leq x_k + P \\ \text{i.e.,} \qquad x_j &\leq x_k + P - T_{setup} \end{aligned}$$

If the right hand side is positive, then x_j can be set to zero without violating any constraints. If not, then $x_k \leq T_{setup} - P < 0$, which implies that k is a flip-flop that will eventually move to the right, thereby allowing flip-flop j the leeway to move as well. Therefore, if this is the case, the skew of flip-flop j is set to

$$x_j = x_k + P - T_{setup} < 0 \tag{18}$$

and the flip-flop j is reprocessed after flip-flop k has been processed (i.e., its skew is set to the value calculated above, and it is placed at the tail of Q). It will be shown in Lemma 8 that all such flip-flops *will* eventually be processed, and their skews set to nearly zero. It is interesting to note that the latter case was almost never seen to happen in the examples presented in this paper.

Note that in spite of Theorem 7, it is still necessary to go through the reverse PERT procedure to ensure that flip-flops are created at fanout points like y_2.

5.3 Minimization procedure for case 2

The steps involved in minimizing the skews at flip-flops with positive skews are analogous to those described in Section 5.2, and are outlined below:

Step 1 All flip-flops in the circuit with positive skews are placed on a queue, Q. Note that each flip-flop has precisely one fanin.

Step 2 Let j be the flip-flop that is currently at the head of the queue, and p the gate that fans into it. As in case 1, we will postpone the discussion of the case where j fans out to a flip-flop. The equivalent skew of the "virtual" flip-flops at every other fanout node of p is found.

The procedure for finding the equivalent skews is as follows. Consider a flip-flop j with positive skew, as shown in Figure 9. The gate p that fans into it to is added to the tail of a queue R.[6] Analogously to the procedure for Case 1, a forward PERT is employed to trace the fanout cone of gate p. When a gate is encountered, it is added to the queue. The trace continues until a flip-flop is encountered. During the process, we keep track of the worst-case delay, d, from gate p. As a consequence of Theorem 1, if the optimal skew at a flip-flop is t units, then its equivalent skew at the output of gate p is $t - d$ units.

[6] As before, note that R is distinct from the queue Q.

Step 3 If any equivalent skew at a "virtual" flip-flop is negative, then the skew at j is set to zero and it is not relocated; if not, the skew after retiming is found using the criteria described earlier. If the magnitude of this skew is smaller than the current skew at j, then j and all of the "virtual" flip-flops at the input to p are retimed across p. The new skews are found as explained in the previous section. Any such flip-flops that have a positive skew (as will happen most of the time, unless relocation changed the sign of the skew) are now placed at the tail of the queue, Q, and are processed later.

Step 4 If the retimed flip-flop has a positive skew, it is placed at the tail of Q.

Step 5 If Q is not empty, go to Step 1.

In Step 2 above, if the flip-flop j fans into another flip-flop, which we shall call k, then since there is no combinational delay between the flip-flops, and retiming preserves the zero-clocking constraints, it must be true that

$$\begin{aligned} x_k + T_{setup} &\leq x_j + P \\ \text{i.e.,} \qquad x_j &\geq x_k - P + T_{setup} \end{aligned}$$

If the right hand side is negative, then x_j can be set to zero without violating any constraints. If not, then we have $x_k \geq P - T_{setup} > 0$, which implies that k is a flip-flop that will eventually move to the left, thereby allowing flip-flop j the leeway to move as well. Therefore, if this is the case, the skew of flip-flop j is set to

$$x_j = x_k - P + T_{setup} > 0 \tag{19}$$

and the flip-flop j is reprocessed after flip-flop k has been processed. It will be shown in Lemma 8 that all such flip-flops *will* eventually be processed. As in the analogous situation in case 1 described in the previous section, it was observed that the latter was almost never seen to happen in the ISCAS89 examples.

6 Properties of the Retiming Procedure

Lemma 8 At the end of the retiming procedure in Phase B, the skew at each flip-flop is no more than half a gate delay.

Proof:

Assume, for purposes of contradiction, that the minimum skew after Phase B at a flip-flop is larger than half a gate delay. Assume also that each flip-flop has a single fanout; in case of multiple fanouts, the flip-flop can be replicated as shown in Figure 8. Then one of two possibilities exist:

(a) If the flip-flop fans out (fans in) to a gate G when the skew is negative (positive), then using the procedure described above, the skew magnitude can be reduced, either to zero, keeping the flip-flop in its current location, or by moving the flip-flop across gate G. This contradicts the fact that Phase B is complete.

(b) If the flip-flop, i, fans out to another flip-flop, j (a primary output is also considered to be a flip-flop), then the skew of flip-flop i is either set to zero (if possible) or to a magnitude that is smaller than that of x_j. This can be seen from Equations (18), (19) and from Lemma 5b (which affirms that $P_{opt} > T_{setup}$).

Now, consider the case of negative skew flip-flops; the situation for positive skew flip-flops is analogous. Assume that we have a situation where flip-flops $j_1, j_2, \cdots, j_n$ are connected directly to each other in a chain and their skews remain negative at the end of Phase B. The flip-flops must necessarily be connected in a cycle; if not, there would be a first (leftmost) flip-flop in the chain that is connected either to a gate or to a flip-flop with zero skew, which implies that their skew magnitudes can be reduced, and contradicts the assumption that Phase B is complete.

Now when each flip-flop in the cycle is processed, the magnitude of its skew is reduced, as shown above. Hence, if the procedure described in Section 5.2 is applied, then each skew magnitude must eventually be reduced to zero, which contradicts the assumption that Phase

B is over. Therefore we cannot have a negative skew flip-flop connected directly to a negative skew flip-flop at the end of Phase B. □

Theorem 9 If, at the end of the retiming procedure, all skews are set to zero, then the optimal clock period for this circuit is no more than $P_{opt} + d_{max}$, where P_{opt} is the optimal clock period found in Phase A, and d_{max} is the maximum delay of any gate in the circuit.

Proof: By Lemma 8, the skew at each flip-flop after Phase B is no more than half a gate delay.

Hence, to ensure that the double-clocking constraint is satisfied, it is easy to see that one may modify the period to

$$\max(P + x_j - x_i) \tag{20}$$

where x_j, x_i are the skews of any pair of flip-flops that are connected by a combinational path. Clearly, since $|x_k| < d_{max}/2 \ \forall \ k$, the clock period required to ensure that all double-clocking constraints are satisfied is no more than $P + d_{max}$. Note also that since any period achievable by retiming is also achievable using skews, and final period can be no better than P, which is the optimal period using skew optimization. □

7 Experimental Results

The algorithm was implemented as a C program, ASTRA. Experimental results running from ASTRA on all circuits in the ISCAS89 benchmark suite (including the Addendum93 circuits) are presented in Table 1. For each circuit, the table provides data that describes its size in terms of the number of combinational gates, $|G|$, and flip-flops $|F|_{init}$. All gates are assumed to have unit delays (although the algorithm is certainly not restricted to unit delay circuits), and the setup and hold times are arbitrarily set to 0. P_{max} is the upper bound on the clock period provided by Lemma 5a. Note that P_{max} corresponds to the clock period in the original circuit.

The next column shows the optimal value, P_{opt}, calculated at the end of Phase A of ASTRA, using clock skew optimization. At the end of retiming in Phase B of ASTRA, any skews that could not be set exactly to zero are now forced to zero and the new period P_{ret} is shown, along with the corresponding percentage improvement over the initial period, P_{max}. This period corresponds to the maximum delay of any combinational segment in the retimed circuit. As expected, it is seen that in each case, P_{ret} is within one gate delay of P_{opt}.

The CPU times for running ASTRA on an HP 735 workstation for each of the two phases, and the total CPU time are shown for all these circuits. Note that the time for Phase A includes the time for calculating the values of $\overline{d}(i, j)$, the maximum delay from flip-flop i to flip-flop j, as well as the time for the Bellman-Ford iterations. The last column shows the number of flip-flops in the retimed circuit.

For example, for s15850.1, a circuit with 9772 gates and 534 flip-flops, the value of P_{max} is found to be 82.0 units. At the end of Phase A (skew optimization), ASTRA calculates the value of P_{opt} to be 63.0 units. The value of P_{ret} at the end of Phase B of ASTRA is 63.0 units. The improvement in the clock period after the two phases is calculated as

$$\%change = \frac{P_{max} - P_{ret}}{P_{ret}} \times 100$$

and is found to be 30.2% for s15850.1. The number of flip-flops was increased in the process from 534 to 572.

It can be seen from the Table that in 36 out of 44 circuits, ASTRA caused the clock period to improve, with the improvement being as much as 220.7% in the case of s6669. Although it is theoretically possible for retiming to reduce the number of flip-flops in the circuit, this was never seen to happen. It should be stressed here that use of the techniques presented here performs the relocation of flip-flops across the minimal number of levels of gates. However, it does *not* minimize the number of flip-flops, and this is a topic for further research.

It is worth noting that the CPU times for ASTRA are rather small; even the largest circuit could be retimed in just over a minute.

The runtimes for the ASTRA algorithm versus the circuit size, $|G|$, is shown in Figure 10. It is very difficult to infer a general relationship between the circuit size and the execution time.

However, it may be inferred from these graphs and from Table 1 that for large circuits where large improvements the clock period are possible, the algorithm is likely to have relatively larger run-times. A second point that can be inferred is that very often, when the clock period is reduced tremendously, the run-time increases because the amount of work required of Phase B increases.

8 Conclusion

An approach that takes advantage of the equivalence between retiming and clock skew is presented, and is used for gate-level retiming. Results on all of the circuits in the ISCAS89 benchmark suite have been presented and can easily be handled by this algorithm.

The chief reason for the improvement is that ASTRA takes a global view of retiming by first solving the clock skew problem in a smaller number of variables. In the second phase, local transformations are used to perform the retiming. The logic behind this approach is that a circuit would have to be very poorly designed indeed to require enormous computation time for the local transformations, and hence in most practical cases, the latter phase takes only a small amount of computation; this is borne out by our experimental results.

It must be pointed out that the algorithm performs retiming only for timing optimization and does not take into account the fact that retiming may cause initial states to change. Therefore, in its current avatar, it is more applicable to the timing optimization of pipelined circuits, rather than for optimization of control unit circuitry, unless such circuits are designed using the techniques in [21].

The ASTRA algorithm may easily be adapted to perform retiming to satisfy a given clock period, P_{spec}. Phase A consists of a single pass through the graph $G_1(P_{spec})$. If P_{spec} is infeasible, this will be reported immediately, and if not, the skew solution at the end of Phase A may be translated to a retiming solution using the methods of Phase B.

The algorithm directly provides a result to the combined clock skew optimization and retiming problem. The use of this algorithm to minimize the number of flip-flops using retiming only is a topic for further research. Another possible extension is the use of deliberate skews with retiming to minimize the number of flip-flops.

Acknowledgments

The authors would like to thank Dr. Jack Fishburn of AT&T Bell Laboratories and Prof. Liang-Fang Chao of Iowa State University for some helpful discussions. They also thank the anonymous reviewers for their helpful comments.

References

[1] C. E. Leiserson and J. B. Saxe, "Retiming synchronous circuitry," *Algorithmica*, vol. 6, pp. 5–35, 1991.

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

[3] H.-G. Martin, "Retiming by combination of relocation and clock delay adjustment," in *Proceedings of the European Design Automation Conference*, pp. 384–389, 1993.

[4] B. Lockyear and C. Ebeling, "Minimizing the effect of clock skew via circuit retiming," Tech. Rep. UW-CSE-93-05-04, Department of Computer Science and Engineering, University of Washington, Seattle, 1993.

[5] L.-F. Chao and E. H.-M. Sha, "Retiming and clock skew for synchronous systems," in *Proceedings of the IEEE International Symposium on Circuits and Systems*, pp. 1.283–1.286, 1994.

[6] E. M. Sentovich *et al.*, "SIS: A system for sequential circuit synthesis," Tech. Rep. UCB/ERL M92/41, Electronics Research Laboratory, University of California at Berkeley, May 1992.

[7] N. Shenoy and R. Rudell, "Efficient implementation of retiming," in *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design*, pp. 226–233, 1994.

[8] S. Chakradhar, "A fast optimal retiming algorithm for sequential circuits," Tech. Rep. 93-C019-4-5506-5, CCRL, NEC USA, Inc, Princeton, NJ, 1993.

[9] 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.

[10] A. Kahng, J. Cong, and G. Robins, "High-performance clock routing based on recursive geometric matching," in *Proceedings of the ACM/IEEE Design Automation Conference*, pp. 322–327, 1991.

[11] R.-S. Tsay, "Exact zero skew," in *Proceedings of the IEEE International Conference on Computer-Aided Design*, pp. 336–339, 1991.

[12] S. Pullela, N. Menezes, and L. T. Pillage, "Reliable nonzero clock skew trees using wire width optimization," in *Proceedings of the ACM/IEEE Design Automation Conference*, pp. 165–170, 1993.

[13] S. Pullela, N. Menezes, J. Omar, and L. T. Pillage, "Skew and delay optimization for reliable buffered clock trees," in *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design*, pp. 556–562, 1993.

[14] L.-F. Chao, *Scheduling and Behavioral Transformations for Parallel Systems.* PhD thesis, Department of Computer Science, Princeton University, 1993.

[15] D. Joy and M. Ciesielski, "Clock period minimization with wave pipelining," *IEEE Transactions on Computer-Aided Design*, vol. 12, pp. 461–472, Apr. 1993.

[16] 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.

[17] T. H. Cormen, C. E. Leiserson, and R. L. Rivest, *Introduction to Algorithms.* New York, New York: McGraw-Hill Book Company, 1990.

[18] S. Even, *Graph Algorithms.* Potomac, MD: Computer Science Press, 1979.

[19] W. Chuang, S. S. Sapatnekar, and I. N. Hajj, "Timing and area optimization for standard cell VLSI circuit design," *IEEE Transactions on Computer-Aided Design*, pp. 308–320, Mar. 1995.

[20] T. G. Szymanski, "Computing optimal clock schedules," in *Proceedings of the ACM/IEEE Design Automation Conference*, pp. 399–404, 1992.

[21] H. J. Touati and R. K. Brayton, "Computing the initial states of retimed circuits," *IEEE Transactions on Computer-Aided Design*, vol. 12, pp. 157–162, Jan. 1993.

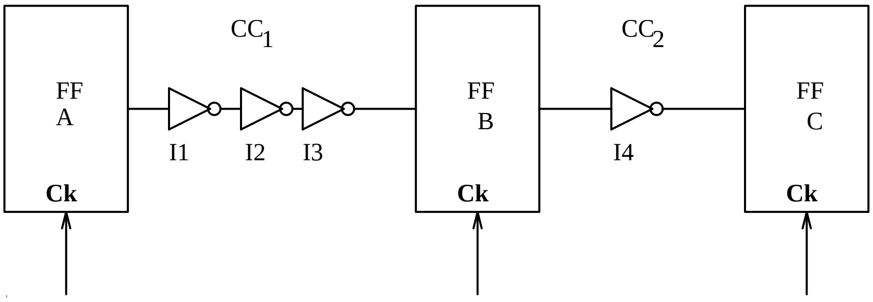

Figure 1: The advantages of nonzero clock skew.

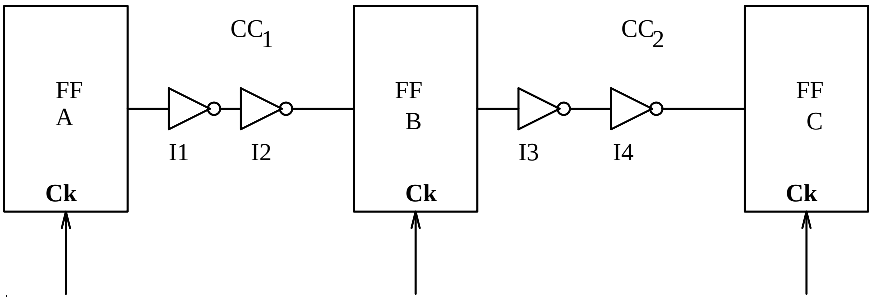

Figure 2: Retiming for clock period optimization.

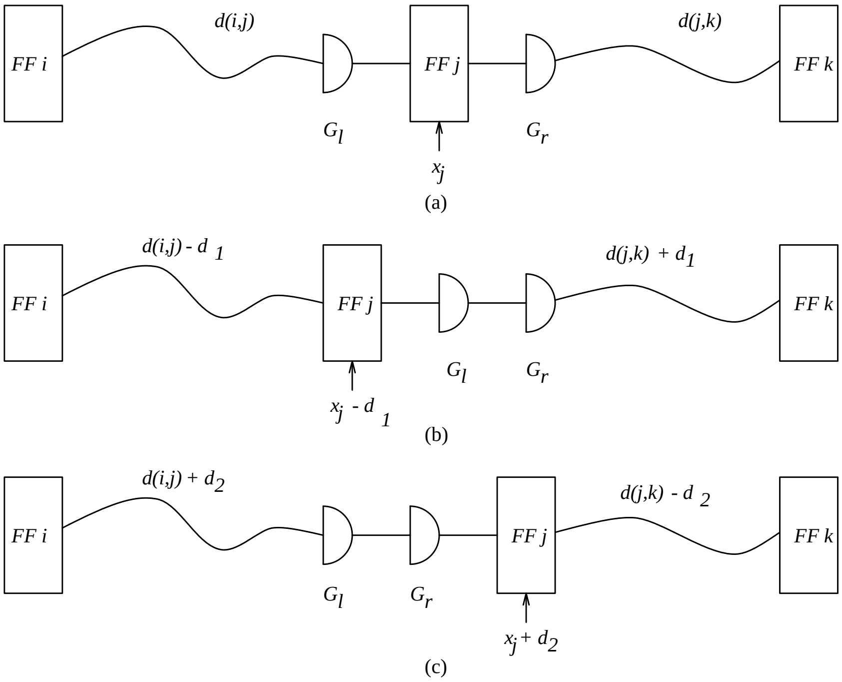

Figure 3: Equivalence between retiming and skew.

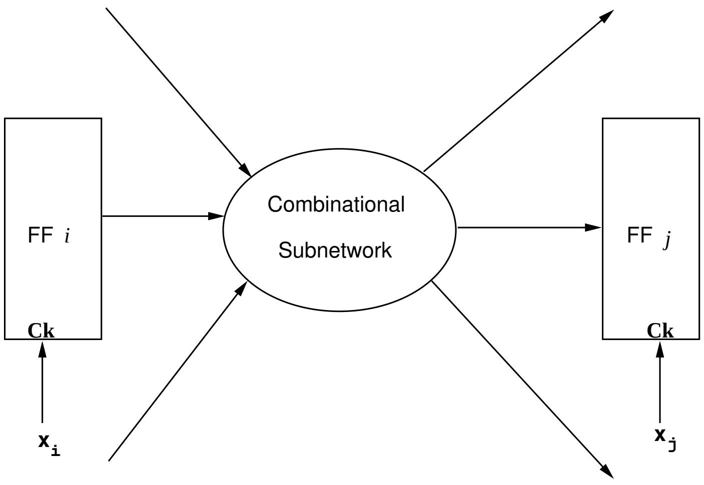

Figure 4: The clock skew optimization problem.

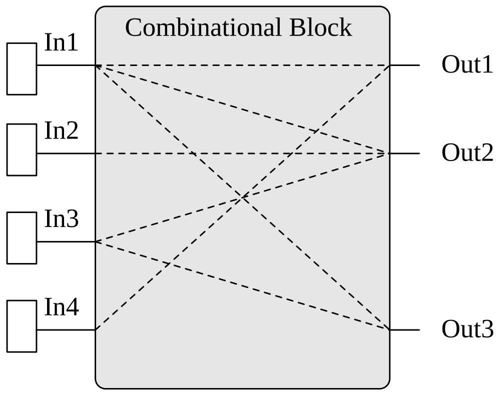

Figure 5: Retiming across a combinational block.

Table 1: Results of Clock Skew Optimization

Circuit	$\|G\|$	$\|F\|_{init}$	P_{max}	P_{opt}	P_{ret}	% change	CPU Time Ph. A	CPU Time Ph. B	CPU Time Total	$\|F\|_{final}$
s27	13	3	6.0	6.0	6.0	0.0%	0.00s	0.00s	0.00s	3
s208.1	112	8	11.0	10.0	10.0	10.0%	0.00s	0.00s	0.00s	27
s298	133	14	9.0	5.3	6.0	50.0%	0.01s	0.01s	0.02s	31
s344	175	15	20.0	14.0	14.0	42.8%	0.01s	0.01s	0.02s	22
s349	176	15	20.0	14.0	14.0	42.8%	0.01s	0.01s	0.02s	22
s382	179	21	9.0	6.3	7.0	28.6%	0.02s	0.01s	0.03s	32
s386	165	6	9.0	6.3	7.0	28.6%	0.01s	0.01s	0.02s	6
s400	175	21	9.0	6.3	7.0	28.6%	0.01s	0.01s	0.02s	32
s420.1	234	16	13.0	12.0	12.0	8.3%	0.02s	0.01s	0.03s	57
s444	202	21	11.0	6.6	7.0	57.1%	0.01s	0.01s	0.02s	56
s499	152	22	12.0	12.0	12.0	0.0%	0.02s	0.01s	0.03s	22
s510	217	6	12.0	11.0	11.0	9.1%	0.01s	0.01s	0.02s	8
s526	214	21	9.0	5.5	6.0	50.0%	0.01s	0.01s	0.02s	40
s526n	215	21	9.0	6.0	6.0	50.0%	0.01s	0.00s	0.01s	36
s635	286	32	127.0	66.0	66.0	92.4%	0.04s	0.02s	0.09s	51
s641	398	19	74.0	74.0	74.0	0.0%	0.04s	0.00s	0.04s	19
s713	412	19	74.0	74.0	74.0	0.0%	0.03s	0.00s	0.03s	19
s820	294	5	10.0	10.0	10.0	0.0%	0.02s	0.00s	0.02s	5
s832	292	5	10.0	10.0	10.0	0.0%	0.02s	0.00s	0.03s	5
s838.1	478	32	17.0	16.0	16.0	6.3%	0.06s	0.03s	0.09s	117
s938	446	32	17.0	16.0	16.0	6.3%	0.06s	0.04s	0.10s	117
s953	424	29	16.0	13.0	13.0	23.1%	0.03s	0.02s	0.06s	41
s967	394	29	14.0	13.0	13.0	7.7%	0.02s	0.01s	0.03s	35
s991	519	19	59.0	57.0	57.0	3.5%	0.06s	0.01s	0.07s	28
s1196	547	18	24.0	24.0	24.0	0.0%	0.03s	0.00s	0.03s	18
s1238	526	18	22.0	22.0	22.0	0.0%	0.04s	0.00s	0.04s	18
s1269	569	37	35.0	18.7	19.0	84.2%	0.07s	0.07s	0.14s	112
s1423	731	74	59.0	53.0	53.0	11.3%	0.13s	0.01s	0.15s	80
s1488	656	6	17.0	16.0	16.0	6.3%	0.04s	0.07s	0.11s	17
s1494	653	6	17.0	16.0	16.0	6.3%	0.04s	0.08s	0.13s	20
s1512	780	57	30.0	24.0	24.0	25.0%	0.05s	0.01s	0.07s	67
prolog	1601	136	26.0	12.5	13.0	100.0%	0.14s	2.63s	2.86s	340
s3271	1572	116	28.0	14.7	15.0	86.7%	0.12s	1.63s	1.88s	428
s3330	1789	132	29.0	14.0	14.0	107.4%	0.14s	2.44s	2.66s	330
s3384	1685	183	60.0	26.5	27.0	122.2%	0.12s	13.9s	15.5s	1697
s4863	2342	104	58.0	30.0	30.0	93.3%	0.24s	1.1s	1.46s	241
s5378	2779	179	25.0	21.0	21.0	19.0%	0.24s	8.0s	8.4s	553
s6669	3080	239	93.0	29.0	29.0	220.7%	0.32s	43.4s	49.3s	2585
s9234.1	5597	211	58.0	38.0	38.0	52.6%	0.8s	9.2s	10.4s	359
s13207.1	7951	638	59.0	51.0	51.0	15.7%	0.98s	0.12s	1.5s	640
s15850.1	9772	534	82.0	63.0	63.0	30.2%	3.31s	2.99s	6.9s	572
s35932	16065	1728	29.0	27.0	27.0	7.4%	3.11s	18.4s	23.1s	1729
s38417	22179	1636	47.0	31.5	32.0	46.9%	56.2s	19.2s	76.9s	1691
s38584.1	19253	1426	56.0	48.0	48.0	16.7%	3.8s	0.4s	5.4s	1428

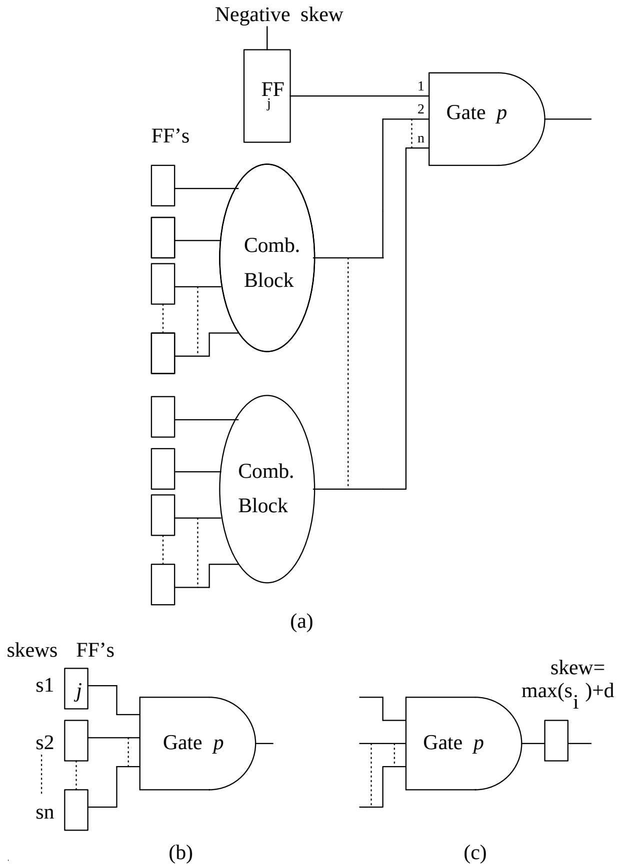

Figure 6: Retiming for a negative skew flip-flop.

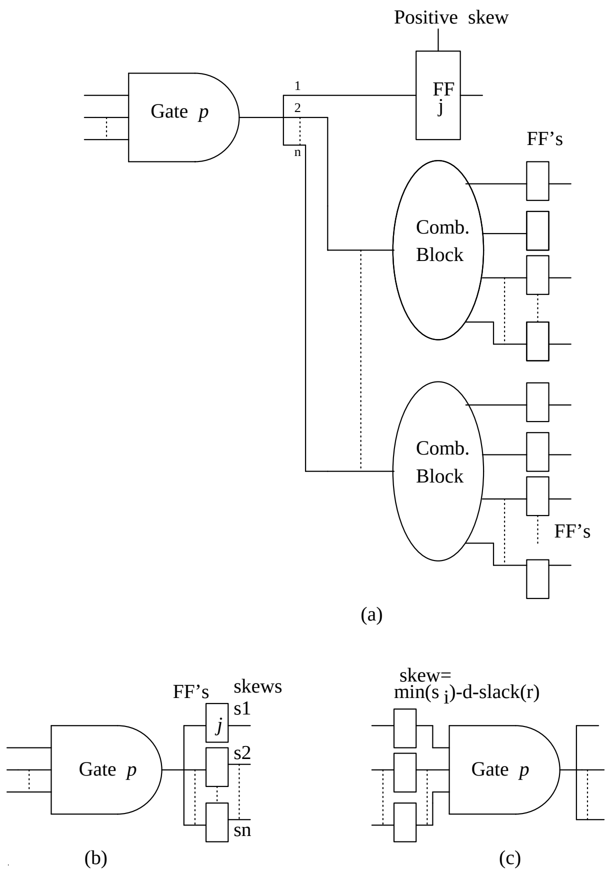

Figure 7: Retiming for a positive skew flip-flop.

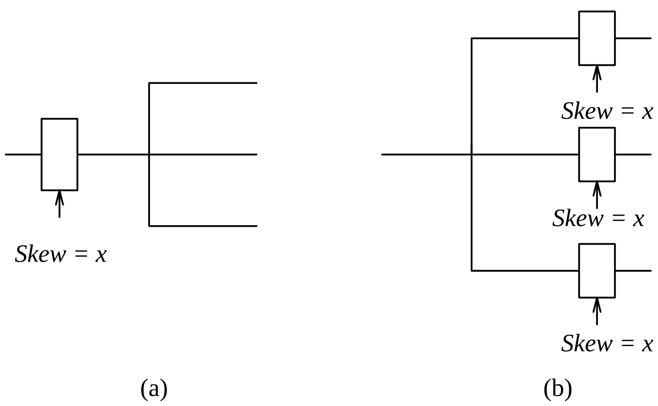

Figure 8: Handling flip-flops with multiple fanouts.

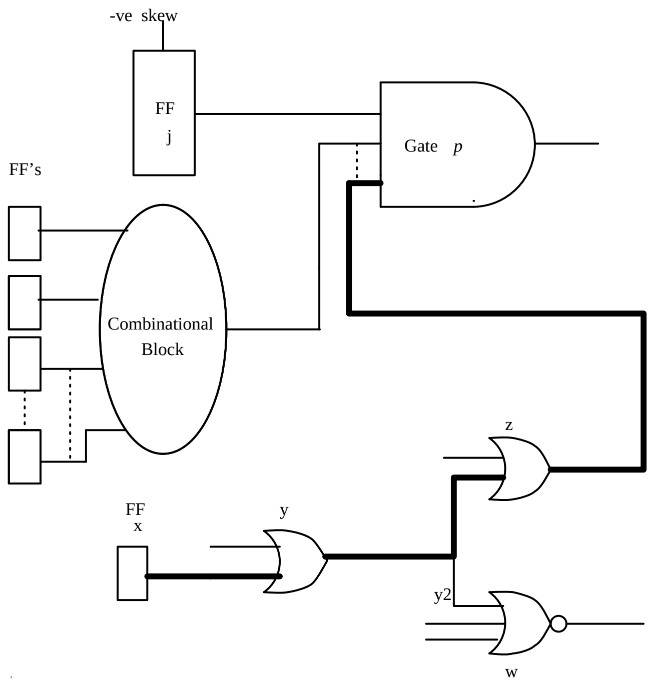

Figure 9: Reproduction of flip-flops in backtrace.

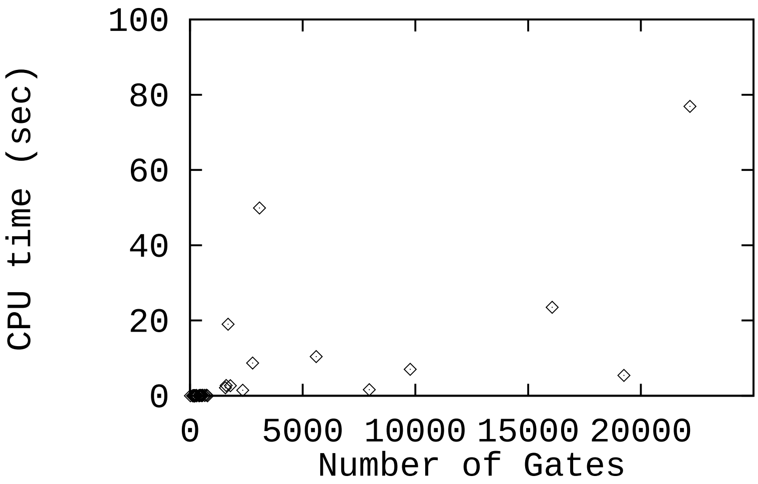

Figure 10: CPU time vs. circuit complexity.