Efficient Minarea Retiming of Large Level-Clocked Circuits

Abstract

Retiming is a powerful transformation that improves performance while preserving functionality. Recent research has led to the development of extremely fast algorithms for retiming edge-triggered circuits, and this paper addresses the harder problem of retiming level-clocked circuits. Level-clocked circuits have the potential to operate faster and require less memory elements than edge-triggered circuits. We present an efficient algorithm for minimum area retiming of large circuits (over 55,000 gates) using multi-phase symmetric clock schedules.

Affiliation

EURO-DAC 97

Category Numbers

Row	Column
04	04
03	04
04	09
03	09

1 Introduction

Timing optimization plays a vital role in the synthesis of VLSI circuits. One method that is of great interest to the design and CAD community is the procedure of retiming, introduced by Leiserson and Saxe [1, 2]. Retiming takes an unoptimized circuit and relocates the memory elements to achieve minimum clock period (minperiod retiming) or minimum area (minarea retiming).

The memory elements in a circuit may be either edge-triggered (called FF's) or level-sensitive (called latches). In a level-clocked circuit the latch is transparent during the active period of the clock. This transparent nature of the latch gives more flexibility both in terms of the minimum clock period achievable and the minimum number of memory elements needed. Unfortunately this transparency also perplexes the analysis of level-clocked circuitry because data can ripple through several stages of memory elements before its propagation is complete. This complicates the design of level-clocked circuits, making the need of good automation tools acute.

As a result several efforts have been made to retime circuits with level-triggered latches based on the Leiserson-Saxe approach, e.g. [3–6]. Since minperiod retiming algorithms [3–5] do no not pay any regard to the number of latches in the retimed circuit, they tend to significantly increase the number of latches. Hence there is a real need for constrained minarea retiming methods. However, the high computational expense of this optimization has limited its use. TIM [6,7] provides one such minarea retiming method for two-phase symmetric clocking schedules. We use the terms "TIM" and "Leiserson-Saxe approach" interchangeably to refer to the Leiserson-Saxe based method to retime level-clocked circuits implemented in TIM.

The starting point of this work is the ASTRA algorithm [8], which presented a different approach to retiming of edge-triggered circuits utilizing the retiming-skew equivalence. This equivalence was extended to level-clocked circuits in [9] which presented an algorithm for minperiod retiming of level-clocked circuits under a general multi-phase clock schedule. Both of these methods are capable of minperiod retiming of large circuits but do not address the harder problem of minarea retiming.

The minarea retiming problem for level-clocked circuits can be formulated as a linear program (LP). This LP is dual of a minimum cost network flow (mincost flow) problem and thus can be efficiently solved [2]. Due to the transparent nature of the latches the number of constraints in this LP is extremely large: for a circuit with $|G|$ gates the number of period constraint is found to be almost $|G|^2/2$. This places heavy time and space requirements on the minarea retiming. Thus TIM is not capable of retiming large circuits for minimum area.

A efficient method for minarea retiming of edge-triggered circuits was presented in [10]. However, the techniques of this work cannot be applied to level-clocked circuits due to the transparent nature of latches. In [11], the concept of bounds on the retiming variables was used to reduce the size of the LP for minarea retiming of edge-triggered circuits. In this work a similar concept is used to solve the much harder problem of minarea retiming of level-clocked circuits. The issues associated with obtaining the bounds, constraint generation and pruning in level-clocked circuits are quite different from the edge-triggered circuits, and the contribution of this work is in the development of fast algorithms for this purpose.

Tight bounds on the variables are obtained by a combination of TIM and ASTRA approaches. In practical circuits the presence of these bounds makes a large fraction of the constraints redundant. This improves the time and space requirements of minarea retiming. By appropriate application of these bounds, we expect not only to prune the constraint set but also to reduce the number of variables. Thus by spending a small amount of additional CPU time on obtaining the bounds this method leads to significant reductions in the total execution time of the minarea retiming for level-clocked circuits, without loss of accuracy. To the best of our knowledge, no published work reduces the size of the LP for minarea retiming of level-clocked circuits. In [5] a technique is suggested for reducing the number of constraints in minperiod retiming, at the cost of some loss in accuracy.

TIM presents results on circuits with less than 400 gates [6]. In this work we show that level-clocked circuit using symmetric clock schedules, with over 56,000 gates can be retimed for minarea in under 1.5 hour. As in all of the references on retiming listed above, this paper assumes the circuit to be composed of gates with constant delays.

The paper is organized as follows. In Section 2 some background material on minarea retiming is presented. Next, in Section 3 we show how to obtain reliable bounds on the retiming variables. Section 4 describes how we generate the reduced LP, the method of solving the LP is then given in Section 5. Experimental results are presented in Section 6, followed by concluding remarks in Section 7.

2 Background

2.1 The Clock Model

A k-phase clock is a set of k periodic signals $\Pi = \langle \phi_1, \gamma_i, \phi_2, \gamma_2, \ldots \phi_k, \gamma_k \rangle$, where ϕ_i is the active duration of phase i and γ_i is the gap between the falling edge of phase i and the rising edge of phase $(i+1)$. We denote the duration of phase i by $\pi_i = \phi_i + \gamma_i$. We overload the symbol Π to also denote the clock period $\Pi = \sum_{i=1}^{k} \pi_i$. A clocking scheme is symmetric if all phases have the same duration and active intervals, i.e. if $\phi_i = \phi \; \forall i = 1 \cdots k$ and $\gamma_i = \gamma \; \forall i = 1 \cdots k$. Thus for a k phase symmetric clocking scheme $\Pi = k \cdot \pi$ and $\pi = \phi + \gamma$. In this work we consider only symmetric clocking schemes.

2.2 The Circuit Model

The graph model used here is identical to the one in [2]. A sequential circuit is represented by a directed graph, $G(V, E)$, where each vertex v corresponds to a gate, and a directed edge e_{uv} represents a connection from the output of gate u to the input of gate v, through zero or more latches. Each vertex has a constant delay $d(v)$. Each edge has associated with it a weight $w(e_{uv})$ and a width $\beta(e_{ij})$. The weight is the number of latches between the output of gate u and the input of gate v. The width of an edge is the area cost of placing one latch on it.

A special vertex called the host vertex (H), is introduced in the graph, with edges from the host vertex to all primary inputs of the circuit, and edges from all primary outputs to the host vertex. To accurately model the number of latches needed in a circuit we take into account maximum latch sharing. To achieve this each gate with multiple fanouts has a mirror gate of zero delay, associated with it; details are given in [2].

A retiming is a labeling of the vertices $r: V \to Z$, where Z is the set of integers. The weight of an edge e_{uv} after retiming, denoted by $w_r(e_{uv})$ is given by

$$w_r(e_{uv}) = r(v) + w(e_{uv}) - r(u) \tag{1}$$

The retiming label $r(v)$ for a vertex v represents the number of latches moved from its output towards its inputs. One may define the weight of any path p originating at vertex u and terminating at vertex v (represented as $u \leadsto v$), $w(p)$, as the sum of the weights on the edges on p, and its delay $d(p)$ as the sum of the weights of the vertices on p. Similarly $w_r(p)$ is the sum of the weights on the edges on p after retiming.

For a level-clocked circuit to be properly clocked the delay between any two gates should be less than the time available, i.e.,

$$d(p) \leq (w_r(p) + 1) \cdot \pi + \phi \tag{2}$$

This constraint can be rewritten after substitution of Equation 1 as

$$r(u) - r(v) \leq w(p) - \frac{d(p)}{\pi} + 1 + \frac{\phi}{\pi} \tag{3}$$

Clearly if there are multiple paths from u to v only the tightest constraint (one with minimum right hand side) is irredundant. We denote the minimum value of $\left[w(p) - \frac{d(p)}{\pi}\right]$ over all paths from u to v by $\delta(u, v)$, i.e.

$$\delta(u, v) = \min_{\forall \ p: u \leadsto v} \left(w(p) - \frac{d(p)}{\pi} \right) \tag{4}$$

Let us define $\Delta(u, v)$ as

$$\Delta(u, v) = \left\lfloor \delta(u, v) + \frac{\phi}{\pi} + 1 \right\rfloor \tag{5}$$

Since the retiming variables $r(u)$ and $r(v)$ are integers, we can rewrite Equation (3) as

$$r(u) - r(v) \leq \Delta(u, v) \tag{6}$$

2.3 The Minarea Retiming Problem

The constrained minarea retiming problem for a target period Π is formulated as the following LP:

$$\begin{array}{lll} \text{minimize} & \sum_{v \in (V)} \left[\left(\sum_{\forall j \in FI(v)} \beta(e_{jv}) - \sum_{\forall j \in FO(v)} \beta(e_{vj}) \right) \cdot r(v) \right] & \\ \text{subject to} & r(u) - r(v) \leq w(e_{uv}) & \forall e_{uv} \in E \\ & r(u) - r(v) \leq \Delta(u, v) & \forall u, v \ \exists \ p: u \leadsto v \end{array} \tag{7}$$

The significance of the objective function and the constraints is as follows (the reader is referred to [2] for details).

- The objective function represents the number of latches added to the retimed circuit in relation to the original circuit taking into account maximal latch sharing at the output of a gate.
- The first set of constraints ensures that the weight e_{uv} of each edge (i.e., the number of latches between the output of gate u and the input of gate v) after retiming is nonnegative. We will refer to this set as the circuit constraint set C_c.
- The second set of constraints ensures that after retiming, each path satisfies the proper timing constraint of Equation (2). This set, being dependent on the clock period, is referred to as the period constraint set C_p.

2.4 Alternate View of Retiming

The ASTRA algorithm in [8] utilized the equivalence between retiming and clock skew optimization to present a alternate view of retiming for edge-triggered circuits. The work in [9] presented a similar equivalence for level-clocked circuits and utilized it for retiming large circuits efficiently. We refer to both of these works as ASTRA.

The details of the ASTRA algorithm are provided in [9]; a brief description is presented here for completeness. The term *Global Departure Time (GDT)* is defined for each latch as the latest departure time of data signal from that latch, with reference to the arrival time at the primary inputs, in a global time frame. GDT is a abstract term and can have either positive or negative values. The problem of finding a set of GDT's which satisfy the target clock period can be easily mapped to the clock skew optimization problem. Since the value of GDT's so obtained may be excessive we need a mechanism for reducing them.

Moving a latch from the inputs of a gate to its outputs is equivalent to increasing the GDT of that latch by the delay of the gate. Likewise, a motion from outputs to the inputs is equivalent to reducing the GDT by the gate delay. A value of GDT between $-\phi$ and 0 is considered physically allowable. Thus relocating latches can be used as a means to bring all the GDT values within this allowable range.

This relationship between GDT and retiming motivates the following two-phase solution to the retiming problem:

Phase A : For a given circuit, one may build a set of difference constraints with one such constraint for every pair of latches that have a purely combinational path connecting them, and these difference constraints may be represented by a constraint graph (notice that this graph is not the same graph as described in the Section 2.2). The Bellman-Ford algorithm [12] may be applied to this graph to find the longest path in the graph. The final values associated with each vertex provide the required GDT at that vertex (latch) that will satisfy the target clock.

Phase B : Latches are relocated across gates in this phase, in an attempt to get all the GDT values to be within allowable range (between $-\phi$ and 0). After Phase B, any GDT value that is not in the allowable range is forced to be in the allowable range. This could cause the clock period obtained in Phase A to increase by a small amount; a upper bound on this increase is given in [9].

3 Restricted Mobility

In this paper, we reconcile the ASTA approach of solving the retiming problem for level-clocked circuits with the TIM approach. We will now show that reliable bounds on the retiming variables r of the TIM approach, can be obtained utilizing the ASTRA approach. These bounds will then be used to reduce the size of the minarea retiming LP in Equation (7).

3.1 The Concept of Restricted Mobility

Consider the simple circuit in Figure 1 with unit gate delays and a single-phase clocking scheme with 50% duty cycle. We assume that the primary inputs and outputs behave like edge-triggered FFs, i.e, that the data signals are available at the primary inputs at the falling edge of the clock, and must arrive at the primary outputs before the falling edge. For any latch that is not a primary input or primary output, the data may depart at any time during the active period of the clock. This assumption gives exactly 2 clock periods to a data signal in Figure 1 to reach the primary output from the primary input.

To achieve the minimum clock period of 2.0 units, one must move the latch $L1$ either to the output of gate $G1$ or to the output of gate $G2$; no other position of the latch $L1$ is feasible. Therefore, it can be seen that the latches cannot be sent to *just any* location in the circuit; rather, there is a restricted range of locations into which a latch may be moved. This restricted mobility of latches will be used to reduce the size of the LP in Equation (7).

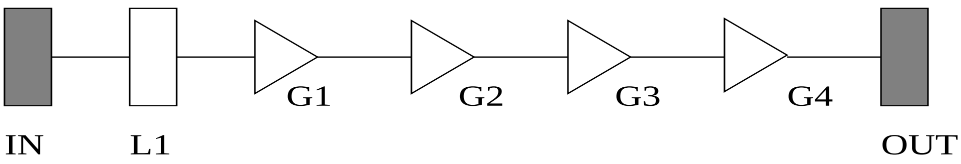

Figure 1: Example circuit.

3.2 The ASAP and ALAP Locations

The concept of restricted mobility is related to the "nearest" and "farthest" location that any latch can occupy under the target clock. These locations can be thought of as the boundaries to the range within which the latch may be moved. Here we use the term ASAP (as-soon-as-possible) locations to refer to the case when all latches are as close to the primary inputs as possible. Similarly the ALAP (as-late-as-possible) locations has all latches as close to the primary outputs as possible[1].

We now describe how a modified form of the minperiod ASTRA [9] can be used to obtain these ASAP and ALAP locations. At the end of Phase A of ASTRA we get *one* possible set of GDT's. Note that this is not the only allowable set of GDT's, since slacks [9] in the arcs of the constraint graph can lead to other allowable solutions. We call the set of GDT values that provide the ASAP locations at the end of Phase B, the ASAP GDT's. The ALAP GDT's are similarly defined.

ASTRA initializes all GDT's to 0 to achieve the minimum range of GDT's. To obtain the bounding GDT's we must initialize all GDT's to $-\infty$. Now when the longest path Bellman-Ford algorithm is applied to the constraint graph for a specified clock, ALAP GDT's are calculated. ASAP GDT's can be obtained by running the Bellman-Ford algorithm on the transpose of this constraint graph [12] (i.e., a graph with the same vertex set as the original graph, but with the edge directions reversed).

These ASAP/ALAP GDT's are then translated to the ASAP/ALAP locations in Phase B of ASTRA. As explained in [9], latches that have positive GDT are moved in the direction opposite to the signal flow direction, and latches with negative GDT are relocated in the direction of signal flow. The procedure of relocating latches to find the ASAP and ALAP locations proceeds along the same lines as in [9], with a few variations described below.

The allowable range (0 to $-\phi$) of GDT's gives us some extra flexibility in the execution of Phase B of ASTRA. If the GDT of a latch is within this allowable range both before and after a possible relocation, the relocation is optional (may or may not be performed). ASTRA uses this flexibility to reduce the total number of relocations made, however; the algorithm for computing the ASAP (ALAP) locations uses this flexibility to keep latches towards the primary inputs (outputs).

When we desire the ASAP locations, the aim is to bring the latches as close to the primary inputs as possible. Therefore, a latch which *can move* in the direction opposite to the signal flow is moved *as much as possible*; while a latch which *must move* in the direction of signal flow is moved *as little as possible*.

Similarly, in finding the ALAP locations the aim is to bring the latches as close to the primary outputs as possible. Therefore, a latch which can move in the direction of the signal flow is moved as much as possible; while a latch which must move in the direction opposite to signal flow is moved as little as possible.

For the example in Figure 1 the ASAP and the ALAP locations are given in Figure 2 and Figure 3 respectively. Notice that a intuitive symmetry is not obtained in the ASAP and ALAP locations due to the assumption that primary inputs and outputs behave like falling edge-triggered FF's.

[1] The calculation of ASAP and ALAP times is a technique that is routinely used in scheduling in high-level synthesis; see, for example, [13].

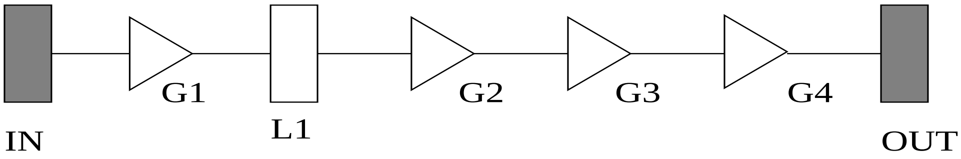

Figure 2: ASAP latch locations after retiming.

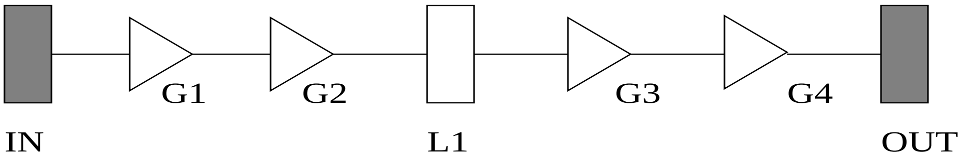

Figure 3: ALAP latch locations after retiming.

3.3 Bounds on the Retiming Variables

During the procedure of obtaining the ASAP and ALAP locations in Phase B of ASTRA, we can also generate the bounds on the r variables. While moving the latches in Phase B, we count the number of latches that traverse each gate. A latch moving from the inputs to the output of a gate decrements the count by one, while one moving from the output to the inputs increments it by one. The count of latches moved across a gate in arriving at the ASAP locations is the upper bound on the r variable for that gate. The lower bound is similarly obtained while retiming for the ALAP locations.

The bounds on the r value of a mirror vertex m_i for the gate i can easily be derived from the bounds on the fanouts of gate i as explained in [11]. To obtain these bounds we utilize (a) the fact from [2] that after optimal retiming the weight on at least one of the edges to the mirror gate will be zero, and (b) the non-negativity constraints on the edges to the mirror gate. The bounds are

$$U_{m_i} = \max_{\forall j \in FO(i)} (U_j + w(e_{ij})) - w(max_i) \tag{8}$$

$$L_{m_i} = \max_{\forall j \in FO(i)} (L_j + w(e_{ij})) - w(max_i) \tag{9}$$

where $w(max_i)$ is the maximum edge weight over all the fanout edges of gate i.

Thus we obtain upper and lower bounds on the r variables corresponding to *each* gate y (including mirror gates) of the form.

$$L_y \leq r(y) \leq U_y \tag{10}$$

We will refer to L_y as the lower bound for gate y and to U_y as the upper bound of gate y. If $U_y = L_y = k_y$ we say that gate y is <u>fixed</u> or immobile since $r(y) = k_y$ is not really a variable any more. On the other hand if $U_y \neq L_y$ we say that gate y is <u>flexible</u> or mobile. The host vertex is assumed to be fixed with $L_H = U_H = 0$. We can now reduce the variable set V of the LP in Equation (7) to $V' \subseteq V$ the variable set of Minaret where

$$V' = \{v \in V | U_v \neq L_v\} \tag{11}$$

Example: For the circuit in Figure 1, the ASAP location for the latch $L1$ is at the output of gate $G1$ as shown in Figure 2. The number of latches moved across each gate in arriving at this ASAP location, and hence the upper bounds are: $U_{G1} = -1$, $U_{G2} = 0$, $U_{G3} = 0$, and $U_{G4} = 0$. The ALAP location of latch $L1$ as shown in Figure 3, is at the output of gate $G2$. The number of latches moved across each gate in arriving at this ALAP location, and hence the lower bounds are: $L_{G1} = -1$, $L_{G2} = -1$, $L_{G3} = 0$, and $L_{G4} = 0$, i.e.

$$\begin{aligned} V &= \{G1, G2, G3, G4\} \\ V' &= \{G2\} \end{aligned}$$

4 Generating the Reduced Linear Program

In this section we will use the bounds on the r variables given in Equation (10) to reduce the LP in Equation (7) to a much smaller LP given below

$$\begin{array}{lll} \text{minimize} & \sum_{v \in V'} \left[\left(\sum_{\forall j \in FI(v)} \beta(e_{jv}) - \sum_{\forall j \in FO(v)} \beta(e_{vj}) \right) \cdot r(v) \right] + K & \\ \text{subject to} & r(u) - r(v) \leq w(e_{uv}) & \forall (u,v) \in C'_c \\ & r(u) - r(v) \leq \Delta(u,v) & \forall (u,v) \in C'_p \\ & L_u \leq r(u) \leq U_u & \forall u \in V' \end{array} \tag{12}$$

where K is a constant accounting for the increase in the number of latches due to fixed gates, the value of which is given by

$$K = \sum_{v \in (V - V')} \left[\left(\sum_{\forall j \in FI(v)} \beta(e_{jv}) - \sum_{\forall j \in FO(v)} \beta(e_{vj}) \right) \cdot U_v \right]$$

The reduced circuit constraint set C'_c is a subset of the circuit constraint set C_c, i.e., $C'_c \subseteq C_c$. Similarly the reduced period circuit constraint set C'_p is a subset of the period constraint set C_p, i.e., $C'_p \subseteq C_p$. The bounds on the variables in V' together with the constant K ensure that the LP's in Equation (7) and Equation (12) have same optimal objective function value. Since K is a constant it can be ignored in the actual optimization process.

4.1 Generating the Reduced Circuit Constraint Set C'_c

We first present a theorem to remove redundant constraints.

Theorem 1 Any constraint (i, j) of the form $r(i) - r(j) \leq c_{ij}$ is redundant in the presence of the bounds of Equation (10) and can be dropped if $U_i - L_j \leq c_{ij}$.

Proof : It can be seen from the bounds on $r(i)$ and $r(j)$ in Equation (10) that

$$r(i) - r(j) \leq U_i - L_j$$

Therefore, if $U_i - L_j \leq c_{ij}$ then $r(i) - r(j) \leq c_{ij}$ must also be true. Thus any constraint (i, j) is redundant and can be dropped if $U_i - L_j \leq c_{ij}$.

Corollary 2 Any constraint (i, j) associated with a fixed node k (i.e. $U_k = L_k$) is redundant in the presence of the bounds of Equation (10) and can be dropped.

Proof : Since both the ASAP and ALAP positions are feasible solutions we are assured that the following holds for all constraints in $C_c \cup C_p$.

$$U_i - U_j \leq c_{ij} \;\; \forall \; (i, j) \in C_c \cup C_p \tag{13}$$

$$L_i - L_j \leq c_{ij} \;\; \forall \; (i, j) \in C_c \cup C_p \tag{14}$$

If a constraint (i, j) is associated with a fixed node then either node i or node j, or both are fixed. If node i is fixed then we have $U_i = L_i$ which along with Equation (14) implies that $U_i - L_j \leq c_{ij}$. If, on the other hand node j is fixed then we have $U_j = L_j$ which along with Equation (13) implies that $U_i - L_j \leq c_{ij}$. In any case Theorem 1 is applicable and the constraint (i, j) is redundant.

The circuit constraint set C_c can be easily obtained by inspection from the circuit. To obtain the reduced circuit constraint set C'_c we accept only those constraints from C_c that are not dropped by the application of Theorem 1. Thus

$$C'_c = \{(i, j) \in C_c | U_i - L_j > w(e_{ij})\} \tag{15}$$

4.2 Generating the Reduced Period Constraint Set C'_p

In order to generate the period constraints we need to compute $\delta(u, v)$ for all-pairs of gates in the circuit. The redundant constraints can then be dropped to obtain C'_p.

4.2.1 Computing the δ Values

The δ values can be obtained by re-weighting each edge e_{ij} to $\left\lceil w(e_{ij}) - \frac{d(i)}{\pi} \right\rceil$ and performing an all-pairs shortest path computation. We use Johnson's algorithm [12] which has $O(|V|)$ memory requirement, since $O(|V|^2)$ memory is not practical for large circuits with tens of thousand gates. Johnson's algorithm first re-weights all edges to ensure nonnegative edge weights. The shortest paths between all pair of gates can then be computed by running Dijkstra's algorithm for each gate as source. However, due to Corollary 2 we need to run Dijkstra's algorithm with only the flexible gates as sources.

We take advantage of the bounds on r variables to speed up the computations. Let us consider a particular run of Dijkstra's algorithm with gate a as the source, and let b be a gate to which the shortest path $\delta(a,b)$ has been obtained. Let c be any other gate in the circuit.

$$\begin{aligned}
\text{By definition} \quad r(a) - r(b) \leq & \quad U_a - L_b \\
\text{If} \quad U_a - L_b \leq & \quad \delta(a,b) \\
\text{then} \quad r(a) - r(b) \leq & \quad \delta(a,b) && (16) \\
\text{From } C'_p \quad r(b) - r(c) \leq & \quad \delta(b,c) + \frac{\phi}{\pi} + 1 && (17) \\
\text{We have} \quad r(a) - r(c) \leq & \quad \delta(a,b) + \delta(b,c) + \frac{\phi}{\pi} + 1 && (18)
\end{aligned}$$

If the shortest path from gate a to gate c does not go through gate b then $\delta(a,b)+\delta(b,c) \geq \delta(a,c)$ and we need not process the fanouts of gate b to obtain $\delta(a,c)$. If on the other hand the shortest path from gate a to gate c is indeed through gate b then $\delta(a,b)+\delta(b,c) = \delta(a,c)$ and Equation (18) is same as the period constraint $r(a)-r(c) \leq \Delta(a,c)$. Since Equation (16) and Equation (17) imply Equation (18), the period constraint (a,c) is redundant if $U_a - L_b \leq \delta(a,b)$. In either case we need not process the fanouts of gate b. Since this is true for any c, we get the following rule.

Rule 1 If during the shortest path calculations from source a using the Dijkstra's algorithm, for any gate b we have $U_a - L_b \leq \delta(a,b)$, we do not need to process the fanouts of gate b.

4.2.2 Reusing δ Computations

We now describe how to reuse some of the computations performed in obtaining the δ values to further speed up the generation of period constraints. We found that in many practical circuits a high percentage of gates have only one fanout. Consider one such gate a having only one fanout b. For the gate a, the shortest paths to all other gates must be via gate b, which means that $\delta(a,c) = \delta(a,b)+\delta(b,c)$. This implies that we can obtain the shortest paths from gate a by simply adding $\delta(a,b)$ to the shortest paths from gate b. Thus if we somehow ensure that shortest paths from gate b are obtained before those from gate a, we will save one complete execution of Dijkstra's algorithm with gate a as source. We call this approach "chaining". We found that, on an average we could reduce the time spent in generating the period constraints by about 50% using a simple chaining technique that required no extra memory. As a side note, Rule 1 must be modified for use with chaining to ensure it is holds for all gates which reuse the δ computation. The idea of chaining can be further generalized. Conceptually there are two extremes of chaining:

- No information about the δ values is stored, e.g., repeated single-source shortest paths algorithms like Johnson's algorithm with $O(|V|)$ memory requirements.
- All information about the δ values is stored, e.g., direct all-pairs shortest path algorithms like Floyd Warshall algorithm [12] with $O(|V|^2)$ memory requirements.

The chaining described above is an intermediate method in which we store δ values from only one source. Conceptually we can define k-chaining as a method that stores δ values from k appropriately chosen sources. This k-chaining in general will require extra memory and careful selection of the k sources and is therefore not considered in this work.

4.2.3 Pruning Redundant Period Constraints

We now present some more techniques to remove redundant period constraints. Consider three gates a, b and c, such that gate b lies on the path from gate a to gate c. Now if

If gate b is a fanin of gate c then we have

$$\begin{aligned} C1: \quad r(a) - r(b) \leq & \quad \Delta(a,b) \\ C2: \quad r(b) - r(c) \leq & \quad w(e_{bc}) \\ C3: \quad r(a) - r(c) \leq & \quad \Delta(a,c) \end{aligned}$$

If $\Delta(a,b) + w(e_{bc}) \leq \Delta(a,c)$ then constraint $C6$ is redundant and can be dropped. This leads us to the following rule

Rule 2 If gate b is a fanin of gate c and $\Delta(a,b) + w(e_{bc}) \leq \Delta(a,c)$ then the period constraint from gate a to c is redundant and can be dropped.

If gate b is a fanout of gate a then we have

$$\begin{aligned} C4: \quad r(a) - r(b) \leq & \quad w(e_{ab}) \\ C5: \quad r(b) - r(c) \leq & \quad \Delta(b,c) \\ C6: \quad r(a) - r(c) \leq & \quad \Delta(a,c) \end{aligned}$$

If $w(e_{ab}) + \Delta(b,c) \leq \Delta(a,c)$ then constraint $C6$ is redundant and can be dropped. This leads us to the following rule.

Rule 3 If gate b is a fanout of gate a and $w(e_{ab}) + \Delta(b,c) \leq \Delta(a,c)$ then the period constraint from gate a to c is redundant and can be dropped.

4.3 Summary

Theorem 1 and Rule 1 are valid only in presence of the bounds and they prune the constraint sets because the information in these bounds make some constraints redundant. Rule 2 and 3 on the other hand do not depend on bounds and, they prune the period constraint set because of the discrete nature of the Δ values. Rules 2 and 3, can be generalized to include implication by more than two constraints; these generalized rules will however, be computationally expensive to apply. This work uses a efficient application of the pruning rules described below, to generate the reduced constraint sets efficiently.

We first use Corollary 2 and Rule 1 to speed up the calculation of the δ values. Next, we use Theorem 1 and Rule 2 to drop redundant constraints. If this sequence is followed it was seen in our experiments with the ISCAS circuits that the application of Rule 3 gives little reduction in the number of period constraints (less than 1%) in the presence of the other rules. Thus to save computation time we do not use Rule 3 in our results.

5 Solving the Linear Program

Like Equation (7), the LP in Equation (12) is also the dual of a mincost flow problem. We applied different min-cost flow methods to solve the problem and found that it could be solved very efficiently using the network simplex algorithm from [14]. The network simplex method is a graph based adaptation of the LP simplex method which exploits the network structure to achieve very good efficiency. The upper and lower bounds on the r variables provide a initial feasible spanning tree. This tree has two levels only, with the host node as the root and all other nodes as leaves. To prevent cycling we construct the initial basis to be strongly feasible by using the appropriate bound (upper or lower) to connect a node to the root (host node). It is easy to maintain strongly feasible trees during the simplex operations, and details are given in [14].

Using the first eligible arc pivot rule with a wraparound arc list from [15] (page 417) gave us significant improvements in the run time. The r variables (duals of the flow variables) are directly available from the mincost flow solution. Using these techniques we could solve mincost flow problems with 70,000 variables and more than 17 Million constraints in about 9 minutes.

6 Experimental Results

Table 1: Quality of Minarea Retiming

Circuit	Π	$\|G\|$	T_{exec}	G_{fx}	F_{avg}	Area		
						Minperiod	Minarea	Edge-triggered
s3384	27.0	1,754	2.71s	10.26%	6.95	685	322	334
s4863	30.0	2,407	5.57s	11.01%	2.38	588	237	276
s5378	21.0	2,863	3.32s	19.66%	2.00	660	314	346
s6669	26.0	3,218	6.69s	8.67%	2.22	1,168	530	610
s9234.1	38.0	3,337	13.36s	10.40%	3.66	590	260	268
s13207.1	51.0	8,004	21.70s	17.57%	6.18	2,831	890	892
s15850.1	63.0	9,843	53.60s	21.60%	3.45	1,718	1,033	1,050
s35932	27.0	16,420	49.51s	21.49%	1.44	4,629	3,457	3,458
s38584.1	48.0	19,595	132.24s	13.78%	4.34	7,399	2,852	2,854
s38417	32.0	21,504	502.97s	0.87%	9.42	6,861	2,722	2,740
myex1	42.0	26,113	551.29s	1.58%	5.66	8,503	4,559	4,586
myex3	94.0	29,134	422.13s	12.63%	4.66	9,288	3,883	3,900
myex2	35.0	35,711	1809.20s	2.37%	5.89	13,309	6,370	6,558
myex4	56.0	40,993	2170.92s	1.19%	9.04	15,087	5,531	5,542
myex5	49.0	56,895	5176.07s	0.67%	9.63	19,307	9,039	9,064

The algorithm was implemented as a C program for minarea retiming, called Minaret. We performed constrained minarea retiming on the complete ISCAS89 benchmark suite. However, due to limitation of space we show results for only the larger circuits. Since we do not have access to large circuits ($> 20,000$ gates) we created some large circuits (myex1 through myex5) by combining circuits from the ISCAS89 benchmark suite. We experimented with both single-phase and two-phase clocking schemes (this work is valid for multi-phase symmetric clocking schemes) but present results only for two-phase clocking schemes. We assume that all gates have a unit delay, although we emphasize that the algorithm is applicable for non-unit delay gates as well.

As in [7] to obtain level-clocked circuits we replaced each edge-triggered FF in the ISCAS89 circuits by two level-sensitive latches. We select the duty cycle and the phase ratio to be 50%,, although this selection is not required by our algorithm.

Table 2: Reduction in the Size of LP in Minaret

Circuit	# Variables			# Constraints			T_b	T_p	T_s
	Minaret	Original	*Red*	Minaret	Original	*Red*			
s3384	1,961	2,166	9.46%	54,216	761,365	92.88%	0.19s	1.95s	0.57s
s4863	2,693	2,995	10.08%	83,752	5,482,144	98.47%	0.15s	4.96s	0.46s
s5378	2,960	3,664	19.21%	31,497	4,595,422	99.31%	0.15s	2.93s	0.25s
s6669	3,787	4,100	7.63%	59,372	1,924,119	96.91%	0.24s	5.99s	0.46s
s9234.1	3,543	3,893	8.99%	136,675	6,443,183	97.88%	0.24s	12.40s	0.72s
s13207.1	7,656	9,180	16.60%	73,796	22,908,612	99.68%	1.05s	19.29s	1.36s
s15850.1	9,013	11,332	20.46%	93,423	39,493,359	99.76%	1.42s	49.27s	2.90s
s35932	17,144	21,716	21.05%	110,019	129,482,466	99.91%	1.85s	42.93s	4.73s
s38584.1	20,590	23,390	11.97%	161,638	293,482,466	99.95%	3.55s	118.21s	10.48s
s38417	25,735	25,923	0.73%	4,390,018	149,498,368	97.06%	4.93s	350.44s	147.60s
myex1	32,509	32,922	1.25%	4,550,735	313,110,465	98.55%	5.59s	484.69s	61.01s
myex3	30,489	34,417	11.41%	213,914	504,058,462	99.96%	7.16s	394.72s	20.25s
myex2	43,837	44,812	2.18%	11,325,336	725,691,302	98.44%	8.16s	1613.56s	187.48s
myex4	48,708	49,214	1.03%	7,400,927	819,727,160	99.10%	10.39s	1926.41s	234.11s
myex5	70,000	70,414	0.59%	17,269,129	1,624,722,880	98.94%	16.84s	4623.62s	535.62s

We present the results in two tables. Table 1 presents the quality of minimum area retiming. For each circuit, the target clock period Π, the number of gates $|G|$, and the CPU time T_{exec} of Minaret is shown. All the execution times are in seconds on a DEC AXP system 3000/900 workstation. The execution time T_{exec} does not include the time spent in reading the circuit, but includes the time spent in calculating the bounds, generating the LP and solving it. Also shown are two metrics on the circuits: G_{fx}, the percentage of gates found to be fixed and F_{avg}, the average flexibility, i.e., the average value of $(U_y - L_y)$ over all gates in the circuit. We also compare the area in terms of the number of latches for minperiod and minarea retiming with the minimum area solution of corresponding edge-triggered circuit. In calculating this area one FF is counted as two latches. The number of memory elements in each case is obtained after taking into account the maximum register sharing [2].

Table 2 shows a closer look at the performance of Minaret on various circuits. The reduction in the number of nodes and constraints over existing methods is shown. We compare the size of the Minaret LP given in Equation (12) and the original LP in Equation (7) by presenting the number of variables and constraints in both. The percentage reduction obtained for Minaret is given as Red. The number of variables include both the gate and mirror variables and hence the reduction in variables Red can be different from G_{fx} which does not include the mirror gates.

Table 2 also presents the breakup of the CPU time in terms of the time spent in deriving the bounds on the r variables (T_b), the time spent in generating the LP (T_p) and the time needed to solve it by the network simplex method described in Section 5 (T_s).

Even though the average flexibility F_{avg} is high and the fraction of fixed gates G_{fx} is low for the circuits we created, we are still able to retime them in very reasonable amount of time. Because of the various pruning techniques used in Minaret the number of constraints in practical circuits grows at a far slower rate than $O(|G|^2)$, which is the rate at which the number of unpruned constraints grow.

Figure 4 shows some area-delay tradeoff curves obtained from ASTRA and Minaret. Since ASTRA pays no attention to minimizing the number of latches, there is no reason for the area-delay curve to be monotonic. The Minaret curve on the other hand must, by definition, be monotonic.

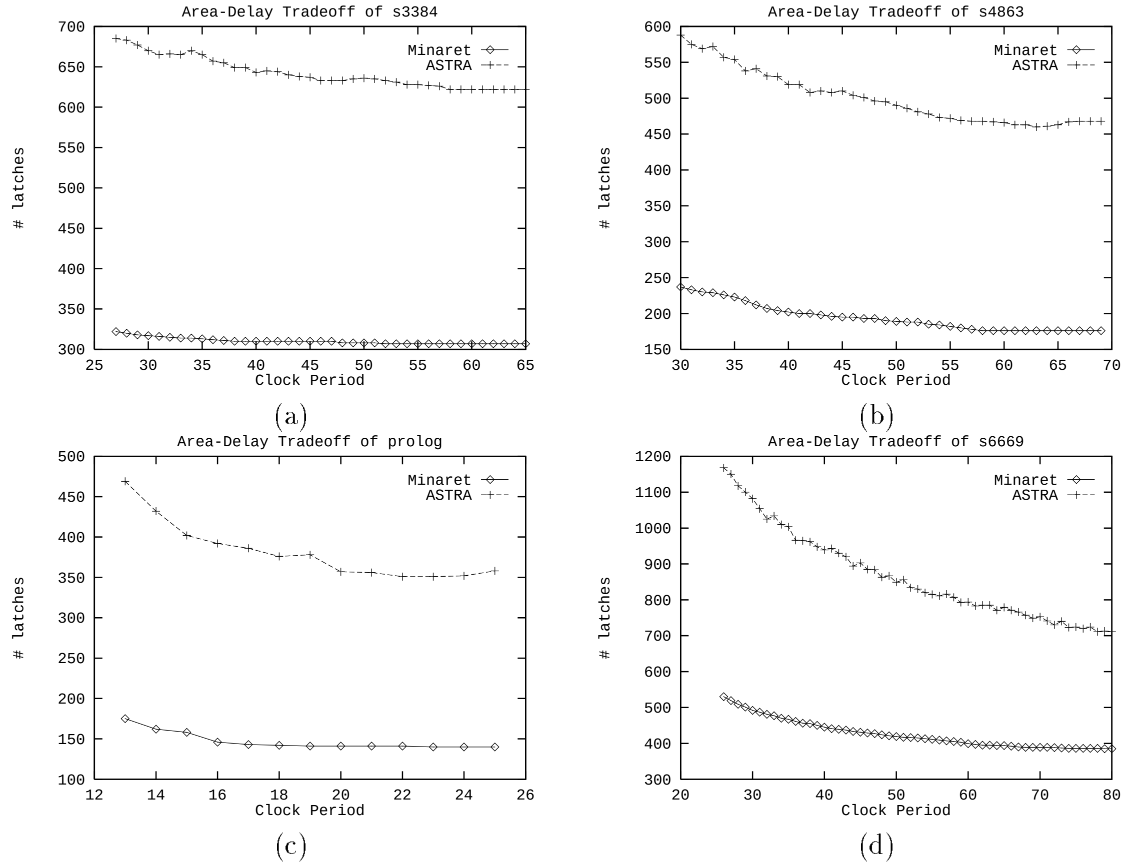

Figure 4: Area-Delay Tradeoff Curves for (a) s3384 (b) s4863 (c) prolog (d) s6669

7 Conclusion

A fast algorithm for minarea retiming of large level-clocked circuits has been presented. The entire ISCAS89 benchmark suite could be retimed in minutes. This algorithm utilizes a modification of the ASTRA method to obtain bounds on the retiming variables. The bounds are then used to efficiently generate a much smaller LP in Equation (12), The optimal objective function value of this LP is same as the original LP in Equation (7), hence no sacrifice in the optimality of the results is made. The contributions of this work are the following:

- It unifies the two approaches to retiming of level-clocked circuits, i.e., the TIM and the ASTRA approaches.
- It provides efficient generation and pruning techniques for the LP. A reduction of two to three orders of magnitudes in the number of constraints is obtained for most circuits.
- It shows that the LP in Equation (12) can be efficiently solved using the network simplex algorithm of Section 5.
- It makes it feasible to retime large level-clocked circuits (over 56,000 gates) for minimum area.

Finally, we note that the concept of using bounds on retiming variables is similar to the one used for minarea retiming of level-clocked circuits in [11]. However, for level-clocked circuits the complexity added by the transparent nature of latches substantially changes the algorithms for the computation of these bounds, the generation of the constraints and their pruning.

References

[1] C. Leiserson, F. Rose, and J. B. Saxe, "Optimizing synchronous circuitry by retiming," in *Proceedings of the 3rd Caltech Conference on VLSI*, pp. 87–116, 1983.

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

[3] N. Shenoy, R. K. Brayton, and A. Sangiovanni-Vincentelli, "Retiming of circuits with single phase transparent latches," in *Proceedings of the IEEE International Conference on Computer Design*, pp. 86–89, 1991.

[4] A. Ishii, C. E. Leiserson, and M. C. Papaefthymiou, "Optimizing two-phase, level-clocked circuitry," in *Advanced Research in VLSI and Parallel Systems: Proceedings of the 1992 Brown/MIT Conference*, pp. 246–264, 1992.

[5] B. Lockyear and C. Ebeling, "Optimal retiming of level-clocked circuits using symmetric clock schedules," *IEEE Transactions on Computer-Aided Design*, pp. 1097–1109, Sept. 1994.

[6] M. C. Papaefthymiou and K. H. Randall, "TIM: a timing package for two-phase, level-clocked circuitry," in *Proceedings of the ACM/IEEE Design Automation Conference*, pp. 497–502, 1993.

[7] M. C. Papaefthymiou, *A Timing Analysis and Optimization System for Level-Clocked Circuitry*. PhD thesis, Massachusetts Institute of Technology, 1993.

[8] S. S. Sapatnekar and R. Deokar, "Utilizing the retiming skew equivalence in a practical algorithm for retiming large circuits," *IEEE Transactions on Computer-Aided Design*, vol. 15, pp. 1237–1248, Oct. 1996.

[9] N. Maheshwari and S. S. Sapatnekar, "A practical algorithm for retiming level-clocked circuits," in *Proceedings of the IEEE International Conference on Computer Design*, pp. 440–445, 1996.

[10] 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.

[11] N. Maheshwari and S. S. Sapatnekar, "An improved algorithm for minimum-area retiming," submitted for publication, 1997.
Manuscript available at `http://www.public.iastate.edu/~naresh/dac97.html`.

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

[13] G. D. Micheli, *Synthesis and Optimization of Digital Circuits*. New York, NY: McGraw-Hill, 1994.

[14] M. S. Bazaraa, J. J. Javis, and H. Sherali, *Linear Programming and Network Flows*. New York, NY: John Wiley, 1977.

[15] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin, *Network Flows Theory, Algorithms and Applications*. Englewood Cliffs, NJ: Prentice Hall, 1993.