Optimizing Large Multi-Phase Level-Clocked Circuits

Naresh Maheshwari and Sachin S. Sapatnekar

Abstract

Retiming is a powerful technique for optimizing sequential circuits. The transparent nature of level sensitive latches enables level-clocked circuits to operate faster and require fewer memory elements than edge-triggered circuits. However, this transparency makes the operation of level-clocked circuits very complex, and optimization of level-clocked circuits is a difficult task. This work presents efficient algorithms for retiming large level-clocked circuits. To provide us with a simpler view of the operation of level-clocked circuits we present the relationship between retiming and clock skew optimization. We then utilize this relationship to develop efficient retiming algorithms for period and area optimization.

For period optimization we present an algorithm which produces near-optimal results, but is significantly faster than the traditional algorithms. In this approach we first calculate the best possible clock period and the amount of motion required for each latch. The latches are then relocated in an attempt to achieve this period. Area, as measured by the number of latches in the circuit can be optimized, by solving a linear program. We apply efficient pruning techniques to reduce the size of this linear program, while preserving optimality. Since generating the linear program is a major part of the computational requirements of minarea retiming, we present techniques for efficient generation of the reduced linear program. This enables us to perform area optimization of large circuits clocked by symmetric multi-phase clocks in very reasonable time, without sacrificing optimality. We present results on circuits with up to 56,000 gates, performing period optimization in under 20 seconds and area optimization in under 1.5 hours.

1 Introduction

Circuit optimization plays a vital role in the design of VLSI systems. Since most systems are sequential in nature, combinational optimization techniques are not able to explore the complete design space. Retiming [1, 2] is a powerful and true sequential circuit optimization technique. Retiming takes an unoptimized circuit and relocates the memory elements to optimize some objective, e.g., clock period [3, 4], area [5, 6], power [7, 8] or testability [9, 10]. The problem of minimizing the clock

[1]N. Maheshwari was with the Department of Electrical and Computer Engineering, Iowa State University, Ames IA 50011. He is now with Synplicity, Sunnyvale, CA. S. S. Sapatnekar is with the Dept. of Electrical and Computer Engineering, University of Minnesota, Minneapolis, MN 55455. This work was supported in part by the National Science Foundation award MIP-9502556, a Lucent Technologies DAC Graduate Scholarship and an Iowa State University Computation Center Grant.

period without regard to the number of latches is called *minperiod retiming*, and the problem of minimizing the number of latches under the constraints of a target clock period is called *minarea retiming*. These two problems are the subject of this work.

Leiserson and Saxe presented polynomial time algorithms for both minperiod and minarea retiming of edge-triggered circuits in [2]. These algorithms were extended to handle level-clocked circuits in [11–14]. Although these algorithms are of polynomial complexity, they are unable to handle large circuits. Recent research [5, 6, 15, 16] has led to the development of fast algorithms for retiming edge-triggered circuits with tens of thousands of gates. In this work we present fast algorithms for retiming level-clocked circuits of similar size in comparable time. Although these methods are very efficient in practice, they do not improve the asymptotic complexity. As in the references on retiming listed above, this paper assumes the circuit to be composed of gates with fixed delays.

The memory elements in a circuit can be either edge-triggered, called *flip-flops (FF's)* or level-sensitive, called *latches*. Unlike an FF, a latch is transparent during the active period of the clock. For edge-triggered circuits (circuits with edge-triggered FF's) the delays through all combinational logic paths must be less than the clock period, hence we must enforce timing constraints only between FF's connected by a purely combinational path. For level-clocked circuits (circuits with level-sensitive latches) the delay through a combinational logic path can be longer than one clock cycle, as long as it is compensated by shorter paths delays in the subsequent cycles. To ensure that the extra delay is compensated we must enforce timing constraints between a latch and every other latch reachable from it (possibly through multiple latches). Consider a linear N stage pipeline with $N+1$ memory elements $(m_0, m_1 \ldots m_N)$. If these memory elements are edge-triggered FF's then we need only N timing constraints ($m_i \rightsquigarrow m_{i+1}$, $0 \leq i \leq N$). However, if these memory elements are level sensitive latches, then we would need $N \cdot (N+1)/2$ timing constraints ($m_i \rightsquigarrow m_j \; \forall j > i$ and $0 \leq i \leq N$). In presence of feedback paths, timing analysis of level-clocked circuits becomes even more complex.

Even though the transparent nature of latches makes design and analysis of level-clocked circuits very complex, they are widely used for high performance designs because they offer more flexibility both in terms of the minimum clock period achievable and the minimum number of memory elements required. Optimizing level-clocked circuits is therefore a complex but important task, and there is an acute need of good automation tools. Several efforts have been made to retime circuits with level-sensitive latches based on the Leiserson-Saxe approach, e.g., [13, 14]. Although these algorithms have polynomial time complexity, their high space and time requirement makes them incapable of handling circuits with even a few thousand gates, and the only published results are on circuits with less than 400 gates. Our goal in this work is to able to retime circuits with tens of thousands of gates in reasonable time, and we present results on circuits with up to 56,000 gates.

These traditional methods [13, 14] solve the minperiod retiming problem by performing a binary search over all possible clock periods. At each step of this binary search, the feasibility of achieving the clock period by retiming is checked by solving a single source shortest path problem using the Bellman-Ford algorithm on a constraint graph. This constraint graph consists of $|G|$ vertices and edges between every pair of vertices (where $|G|$ is the number of gates in the circuit), and is

obtained by solving an all-pairs shortest path problem on the original circuit graph. This graph has to be reconstructed for every binary search point, because as shown in [14, Section VI-A], unlike edge-triggered circuits, critical paths in level-clocked circuits can be different for different clock periods. Therefore [13] and [14] have $O(|G|^2)$ space requirement and high (although polynomial) time complexity. This complexity of retiming level-clocked circuits arises due to the transparent nature of latches, which forces us to consider constraints on paths going through multiple latches.

In this work our goal is not to reduce the asymptotic complexity of retiming level-clocked circuits, but to reduce the run time in practice. We present a bounded-approximation algorithm for minperiod retiming of very large multi-phase circuits with general clock schedules. This is achieved by introducing the concept of *Global Departure Time (GDT)* to map the minperiod retiming problem to a skew optimization problem and thus solving it much like the simpler problem of retiming edge-triggered circuits using the approach of [15]. In each step of the binary search we solve the single source shortest path problem on a much smaller constraint graph with only $|\Psi|$ vertices, where $|\Psi|$ is the number of latches in the circuit. This constraint graph contains edges only between latches that have a purely combinational path between them, and therefore in practice is much smaller and sparse as compared to the constraint graph in traditional methods. Unlike the traditional methods that reconstruct the constraint graph for every binary search point, we perform a simple reweighting of the edges. Once the minimum period is obtained, the latches are relocated to obtain this minimum period.

We also present a practically efficient yet optimal algorithm for minarea retiming of level-clocked circuits. The minarea retiming problem can be formulated as a linear program (LP) [2]. The work in [2] is restricted to edge-triggered circuits, and the work in [13, 14, 25] extends this approach to handle the problem of retiming level-clocked circuits with symmetric clocking schemes. This LP is generated by solving an all-pair shortest path problem, and has $|G|$ variables and almost $\frac{|G|^2}{2}$ constraints. This LP can be solved efficiently by solving its min-cost flow dual [2]. For edge-triggered circuits, the work in [5] presented an technique for pruning the number of constraints, which utilized the observation that in edge-triggered circuits, if a subpath satisfies the timing constraints, then any path containing this subpath will also satisfy the timing constraints (unfortunately this is not true for level-clocked circuits due to the transparent nature of latches). The work in [6] built on the idea and added efficient techniques to obtain bounds on the variables of the LP for edge-triggered circuits. These bounds were used to further reduce the size of the LP and the time required to generate it.

The concept of GDT presented in this work makes it possible for us to apply similar techniques to generate bounds on the variables in the minarea LP for level-clocked circuits, and to use it to reduce the size of this LP. However, due to the transparent nature of latches, unlike edge-triggered circuits, the techniques of [5] and [6] cannot be used to reduce the time required to generate the minarea LP in level-clocked circuits. This presents a major hurdle in retiming large level-clocked circuits for minimum area, because in the absence of any efficiency-improving techniques, the minarea LP can not be generated in any reasonable time. In this work we present new techniques for pruning the minarea LP for level-clocked circuits, and reducing the time required to generate it. These

new techniques reduce the number of constraints by a factor of 20 to 40, and also reduce the time required to generate these constraints by a factor of 10 to 20. These reductions are in addition to those obtained by a simple extension of the techniques in [6]. Using the techniques presented in this paper the entire ISCAS-89 benchmark suite could be retimed for minimum period in seconds, and for minimum area in minutes.

The remainder of the paper is organized as follows. In Section 2, we present some background material, after which in Section 3, we discuss a relation between retiming and clock skew optimization for level-clocked circuits. This relation is then utilized for efficient minimum period and minimum area retiming in Section 4 and Section 5 respectively. Experimental results are presented in Section 6, followed by concluding remarks in Section 7.

2 Background

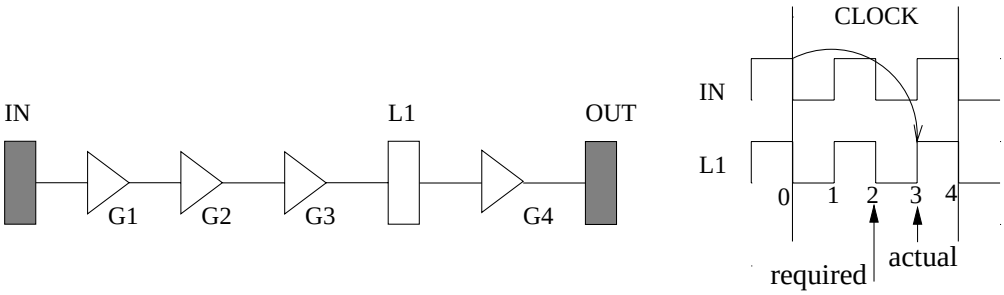

Figure 1: An example circuit.

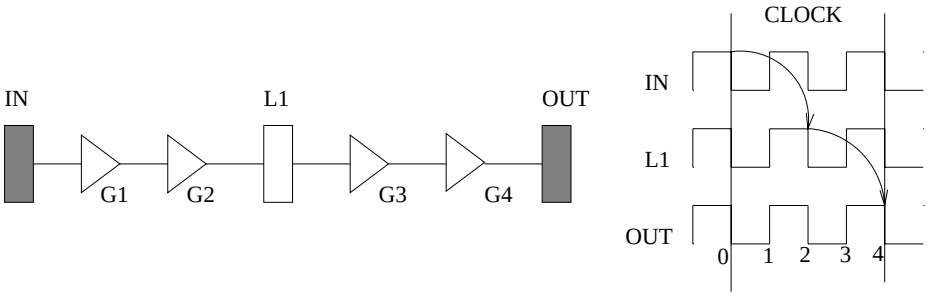

Figure 2: Retiming for clock period optimization.

Consider the simple circuit in Figure 1 with unit delay gates and a single-phase clocking scheme with 50% duty cycle. In this work we will assume 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. Under this assumption a data signal in this circuit gets exactly two clock periods to reach the primary output from the primary input.

A clock period of 2.0 units is not feasible for the circuit in Figure 1. This is because as shown in the figure the actual arrival time (3.0 units) is one time unit later than the required arrival time (2.0

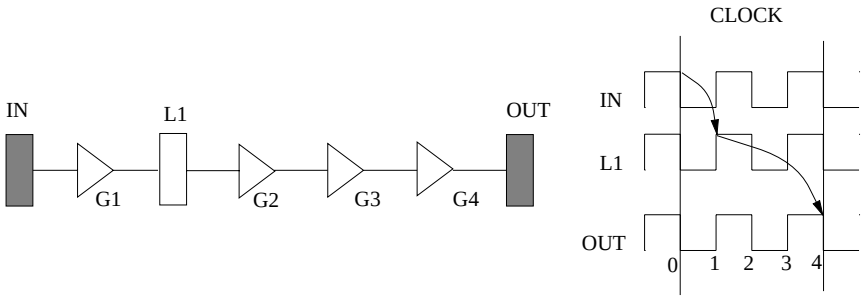

Figure 3: Alternate retiming for clock period optimization.

units). Hence the minimum clock period at which this circuit can operate without any modifications is 3.0 units. However, a clock period of 2.0 units can be achieved by moving the latch L1 across the gate G3. Notice that this is not the only possible location of memory element L1 that can achieve the clock period of 2.0 units; placing latch L1 at the output of gate G1 also achieves the same clock period as shown in Figure 3. This is possible because of the transparent nature of the latches which allows the data signal to depart from the latch at any time during the active period of the clock.

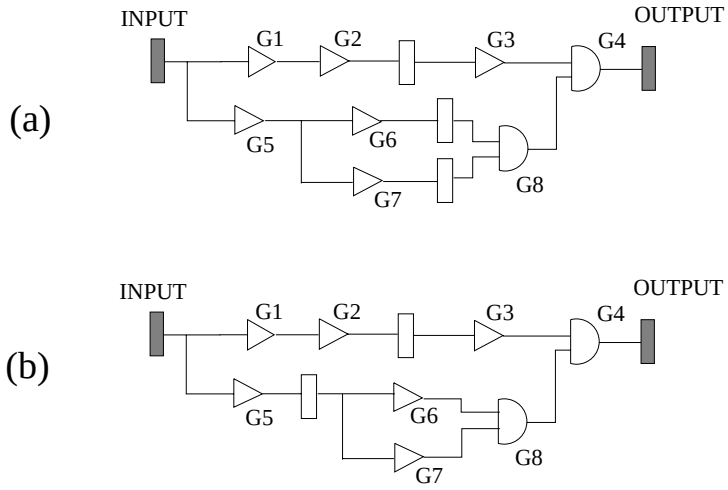

Figure 4: An example of minimum area retiming.

Relocating latches can also reduce the number of latches in a circuit. Consider the circuit in Figure 4(a), with unit delay gates, three latches, and a clock period of 2.0 units. By moving the latches at the outputs of gate G6 and G7 to the output of gate G5, as shown in Figure 4(b), we can reduce the number of latches from three to two, without changing the clock period of the circuit. Notice that this reduction in the number of latches would not have been possible with edge-triggered FF's without increasing the clock period to 3.0 units.

Thus latches can be moved (retimed) across gates to either reduce the clock period or the number of latches in a circuit. We use the term *right* to denote the direction of the signal flow and *left* to indicate the direction against the signal flow. Thus retiming a latch by moving it to the right across a gate implies removing a latch from each of the fanins of that gate and adding one to all of the fanouts of that gate. Similarly retiming a latch left across a gate implies removing a latch from each

of its fanouts and adding one to each of the fanins.

2.1 Circuit Model

As in [2], we represent the circuit by a directed graph, $G(V, E)$, where each vertex $v \in V$ corresponds to a gate in the circuit, 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 v has a fixed delay $d(v)$, and the fanin and fanout set of vertex v is denoted by $FI(v)$ and $FO(v)$ respectively. A special vertex called the host vertex (H) with zero delay 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. Each edge has associated with it a weight $w(e_{uv})$, and a width $\beta(e_{uv})$. The weight $w(e_{uv})$ is the number of latches between the output of gate u and the input of gate v, while the width $\beta(e_{uv})$ of an edge is the area cost of placing one latch on it. The set of latches in the given circuit is denoted by Ψ.

Retiming is a labeling of the vertices $r : V \rightarrow Z$, where Z is the set of integers. The retiming label $r(v)$ for a vertex v, also referred to as a lag, represents the number of latches moved from its output towards its inputs. The weight of an edge e_{uv} after retiming, denoted by $w_r(e_{uv})$ is given by $w_r(e_{uv}) = r(v) + w(e_{uv}) - r(u)$. One may define the weight $w(p)$ of any path $p : u \rightsquigarrow v$, originating at vertex u and terminating at vertex v as the sum of the weights on the edges on p, and its delay $d(p)$ as the sum of the delays of the vertices on p. Similarly $w_r(p)$ is the sum of the weights on the edges on p after retiming, and is given by

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

2.2 Clock Model

In this work, we have adopted the clock model of Sakallah, Mudge and Olukotun [17], and we describe it here for completeness. A k-phase clock is a set of k periodic signals, $\Phi = \{\phi_1 \ldots \phi_k\}$ where ϕ_i is referred to as phase i of the clock Φ. All of the ϕ_i's have the same clock period T_Φ, and each phase i has an active interval of duration T_{ϕ_i} and a passive interval of duration $(T_\Phi - T_{\phi_i})$. Each latch $i \in \Psi$ is clocked by exactly one phase of the clock Φ, which is denoted by $p(i)$. The latches controlled by a clock phase are enabled during the active interval and disabled during the passive interval. When the clock period, T_Φ, is changed, the active intervals of each phase are scaled proportionately. The term "clocking scheme" is used to indicate the relative ratios and duty cycles of the individual phases. Thus a clocking scheme together with a clock period T_Φ, defines a "clock schedule" Φ.

Associated with each phase i is a *local time zone*, shown in Figure 5, such that the passive interval starts at time 0, the enabling edge occurs at time $(T_\Phi - T_{\phi_i})$, and the latching edge occurs at time T_Φ. There is also a global time reference and e_i denotes the time when the phase ϕ_i ends, relative to this global time reference. Phases are ordered so that $e_1 \leq e_2 \ldots \leq e_{k-1} \leq e_k = T_\Phi$, and are numbered modulo-k, i.e., $\phi_{k+1} = \phi_1$ and $\phi_{1-1} = \phi_k$.

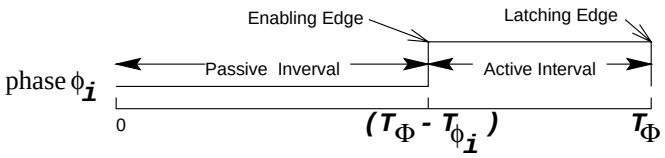

Figure 5: Phase i of a k-phase clock (all times in local time zone).

A phase shift operator $E_{i,j}$, shown in Figure 6, is defined as follows:

$$E_{i,j} = \begin{cases} (e_j - e_i) & \text{for } i < j \\ (T_\Phi + e_j - e_i) & \text{for } i \geq j \end{cases} \tag{2}$$

Note that $E_{i,j}$ takes on positive values, and when subtracted from a time point in the current time zone of ϕ_i, it changes the frame of reference to the next local time zone of ϕ_j.

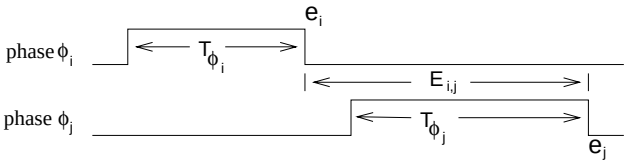

Figure 6: The phase shift operator.

2.3 Timing Constraints for Level-Clocked Circuits

We now enumerate the set of timing constraints, that dictate the correct operation of a level-clocked circuit. We neglect to consider latch setup and hold times here, since they can be incorporated easily by including the setup times in the path delays and the hold time in the clock periods.

Each latch i also has an associated latest arrival time A_i, and a latest departure time D_i, in its local time zone. Due to the transparent nature of the latches, a signal can depart from a latch i any time during the active interval of the phase $p(i)$, i.e.,

$$T_\Phi - T_{\phi_{p(i)}} \leq D_i \leq T_\Phi.$$

However, a signal cannot depart from a latch before it has arrived at that latch, i.e.,

$$A_i \leq D_i.$$

The arrival time at a latch j of a signal departing from another latch i connected by a purely combinational path (denoted as $i \hookrightarrow j$) of delay d_{ij} must satisfy the following relation

$$D_i + d_{ij} - E_{p(i),p(j)}) \leq A_j.$$

Combining the above relations we can obtain the timing constraints for proper clocking of level

clocked circuits, considering only long path constraints[1] as

$$\begin{aligned} D_i + d_{ij} - E_{p(i),p(j)} &\leq D_j \quad \forall\, i \hookrightarrow j \mid i,j \in \Psi \\ T_\Phi - T_{\phi_{p(i)}} &\leq D_i \leq T_\Phi \quad \forall\, i \in \Psi \end{aligned} \tag{3}$$

3 Relation Between Retiming and Skew

Clock skew at any latch is defined as the time by which the clock is delayed in arriving at the latch, with respect to a fixed reference (the arrival time of the clock at the primary inputs). Clock skews have traditionally been considered to be a liability and various techniques to get a skew-free clocking network have been proposed [19–21]. An alternative approach views clock skews as a manageable resource rather than a liability, and intentionally introduces skews to improve the performance of the circuit [22]. Consider the circuit in Figure 1 where the clock period of 2.0 units is not feasible since the actual arrival time (3.0 units) is one time unit after the required arrival time (2.0 units). However, as shown in Figure 7 if a skew of +1.0 unit is applied to the clock at latch L1, the required arrival time at latch L1 becomes 3.0 time units, and the data is properly clocked at latch L1. The circuit can now run with a clock period of 2.0 units. Thus clock skews can be used to improve the performance of a circuit.

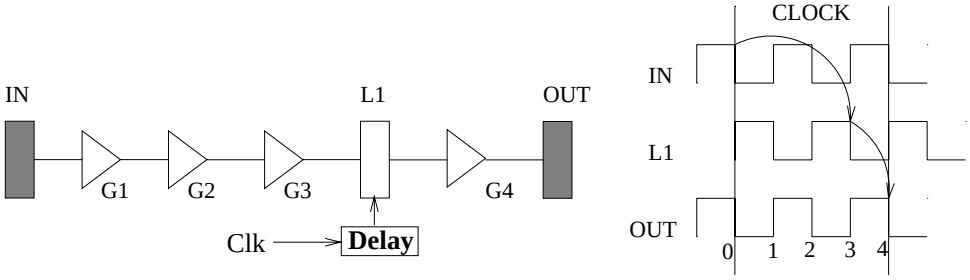

Figure 7: Using clock skew to reduce clock period.

To derive timing constraints in presence of skews we now augment the Sakallah-Mudge-Olukotun model with our own notation. We associate a skew S_i with every latch $i \in \Psi$. Note that the skew values here are not physical skews to be applied to the final circuit, but conceptual ideas that will eventually help us to achieve a retiming solution. Therefore no restrictions are placed on the value of S_i, i.e. $-\infty \leq S_i \leq \infty$.

We define a *latch shift* operator $L_{i,j}$, shown in Figure 8, much like the phase shift operator. This operator converts time from the local time zone of latch i to the local time zone of latch j, taking into account their skews. It is defined as

$$L_{i,j} = \begin{cases} (S_j + e_{p(j)}) - (S_i + e_{p(i)}) & \text{for } i < j \\ T_\Phi + (S_j + e_{p(j)}) - (S_i + e_{p(i)}) & \text{for } i \geq j \end{cases}$$

[1] We do not consider short path constraints here, and rely on techniques such as min-padding [18] to correct any short path violations.

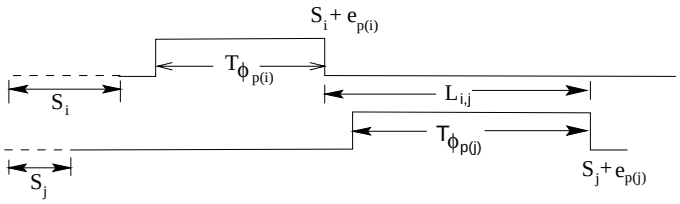

Figure 8: The latch shift operator.

which can be rewritten in terms of the phase shift operator as

$$L_{i,j} = (S_j - S_i) + E_{p(i),p(j)} \tag{4}$$

In presence of skews at latches, the timing constraints in inequality (3) must be modified by using the latch shift operator instead of the phase shift operator. Thus the timing constraints for a level clocked circuit to be properly clocked by a clock schedule Φ, in presence of skews are

$$\begin{aligned} D_i + d_{ij} - L_{i,j} &\leq D_j \quad \forall\, i \hookrightarrow j \mid i,j \in \Psi \\ T_\Phi - T_{\phi_{p(i)}} \leq D_i &\leq T_\Phi \quad \forall\, i \in \Psi \end{aligned}$$

These timing constraints can be rewritten as

$$\begin{aligned} (S_i + D_i) + d_{ij} - E_{p(i),p(j)} &\leq (S_j + D_j) \quad \forall\, i \hookrightarrow j \mid i,j \in \Psi \\ T_\Phi - T_{\phi_{p(i)}} &\leq D_i \leq T_\Phi \quad \forall\, i \in \Psi \\ -\infty &\leq S_i \leq \infty \quad \forall\, i \in \Psi \end{aligned}$$

To make the discussion simpler we subtract T_Φ from both sides of the first relation, and substitute

$$X_i = (S_i + D_i - T_\Phi) \tag{5}$$

We refer to X_i as the *Global Departure Time* (GDT). This gives us

$$\begin{aligned} X_i + d_{ij} - E_{p(i),P(j)} &\leq X_j \quad \forall\, i \hookrightarrow j \mid i,j \in \Psi \\ -\infty \leq X_i &\leq \infty \quad \forall\, i \in \Psi \end{aligned}$$

These can be written as the following set of difference constraints.

$$\begin{aligned} X_i - X_j &\leq E_{p(i),p(j)} - d_{ij} \quad \forall\, i \hookrightarrow j \mid i,j \in \Psi \\ -\infty \leq X_i &\leq \infty \quad \forall\, i \in \Psi \end{aligned} \tag{6}$$

As shown earlier, both skew and retiming can modify the circuit in Figure 1 to operate at a faster clock period of 2.0 units. In fact both achieve this objective by the same basic principle of borrowing time from one cycle and lending it to another. Therefore retiming and skew optimization can be

considered to be related to each other. A formal presentation of this relationship is given in [15], for edge triggered FF's. An FF can be conceptualized as a level sensitive latch with a very small active interval.

The physical meaning of GDT is as follows. If we can apply arbitrary skews at latches, we can adjust the skew, S_i, of a latch so as to force $D_i = T_\Phi$, which is same as a negative edge triggered FF. Since $X_i = S_i + D_i - T_\Phi$, setting $D_i = T_\Phi$ gives $S_i = X_i$. Hence, we can look at X_i for latches in the same way as we look at skews for FF's.

The difference constraint between GDT values of two latches given in inequality (6) is similar to the difference constraints between skews at FF's in [15]. Therefore, we suggest a relation between retiming and GDT values of level-sensitive latches, similar to the one given in [15] between retiming and skews for edge triggered FF's. This relation will allow us to develop efficient techniques for retiming level-clocked circuits. We now state a theorem similar to Theorem 1 in [15]; the proof of this theorem is similar to the one given in [15].

Theorem 1 *In a level-clocked circuit, retiming a latch across a gate with delay d_1 by applying to it a positive [negative] lag is equivalent to increasing [decreasing] its GDT by d_1.*

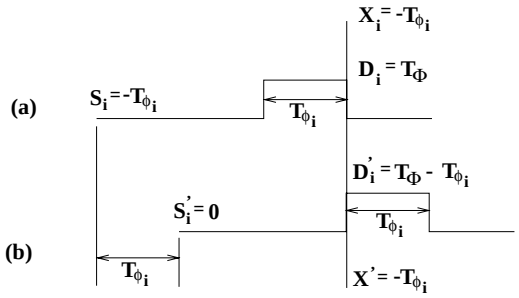

Figure 9: The ability of a latch to absorb some skew.

Note that in reality, we are not compelled to set $D_i = T_\Phi$, and that we can reduce D_i by as much as T_{ϕ_i} and increase S_i by the same amount, while keeping X_i constant. Since only GDT's (X_i's) appear in the timing constraint of relation (6), keeping them constant keeps the clock period constant. Consider Figure 9 (a) where $S_i = -T_{\phi_i}$ and $D_i = T_\Phi$, thus $X_i = -T_{\phi_i}$. We can increase the skew to zero ($S'_i = 0$), without changing the GDT as shown in Figure 9 (b), by reducing the departure time by the same amount $D'_i = T_\Phi - T_{\phi_i}$, leaving the GDT unchanged ($X'_i = X_i = -T_{\phi_i}$). Therefore, we can absorb a skew of up to $-T_{\phi_i}$ in the D_i without violating the long path constraint. Thus a GDT value between $-T_{\phi_i}$ and 0 is allowed and this range will be referred to as the allowable range. If different phases have different active interval then this allowable GDT range of a latch will depend on its phase. Therefore in our model, level-sensitive latches can be conceptualized as FF's that have the capacity to absorb some skew.

At this time, we also note the relation between the GDT, X_i, of a latch i and the corresponding

minimum magnitude skew, S_i:

$$S_i = \begin{cases} X_i & \text{if } X_i \geq 0 \\ 0 & \text{if } -T_{\phi_i} \leq X_i \leq 0 \\ X_i + T_{\phi_i} & \text{if } -T_{\phi_i} > X_i \end{cases} \tag{7}$$

4 Minimum Period Retiming

Given a circuit and a clocking scheme, minimum period (minperiod) retiming finds the minimum possible clock period T_Φ, for which there exists a retimed circuit that will be properly clocked by Φ (the clock schedule for the given clocking scheme and clock period T_Φ), and the retiming that makes this clock schedule possible. As mentioned in Section 1, the traditional techniques of [13, 14] are unable to handle large level-clocked circuits in a reasonable time. We utilize the relationship between GDT and retiming presented in Section 3 to map the problem of retiming level-clocked circuits for minimum period to the simpler problem of retiming edge-triggered circuits for minimum period as solved in [15]. This mapping motivates the following two-phase method of retiming for minimum clock period under a given clocking scheme.

Phase A : Find the minimum clock period and a set of GDT values that will achieve this period.

Phase B : Relocate latches across gates in an attempt to get all the GDT values to be within allowable range.

As mentioned later in Section 4.1, in Phase A of this method we construct a potentially small and sparse graph only once, unlike the traditional methods [13, 14] which construct multiple large and dense graphs. In Phase B we perform fast local transforms to obtain the retiming solution. Therefore using this two phase method we can retime large level-clocked circuits very efficiently.

As in [15] it must be noted that since gate delays take on discrete values, it cannot be guaranteed that the GDT at every latch can be reduced to be within the allowable range through retiming operations. Thus it is possible that like ASTRA [15] our method may not be achieve the optimal retiming clock period. After the GDT values have been reduced as much as possible, the retimed circuit may be implemented either by applying the requisite (remaining) skews at a latch (to get the optimal clock period achievable by skew optimization), or by setting all skews to zero to get a clock period that is, as will be shown in Section 4.4, no more than a fixed bound above the optimal period with skews. Note, however, that this is not necessarily suboptimal since the minimum clock period using skews may not be achievable using retiming alone, since retiming allows cycle-borrowing only in discrete amounts (corresponding to gate delays), while skew is a continuous cycle-borrowing optimization [22]. As will be shown in Section 4.4, if the maximum gate delay is less than the least T_{ϕ_i}, we can always achieve the optimal skew optimization period by retiming alone.

We first describe the two phases of minperiod retiming, followed by the special case of retiming a circuit for a given clock period. We then present the bound on the difference between the optimal skew optimization period and the clock period obtained by our method.

4.1 Phase A: Clock Period Optimization

The problem of minimizing the clock period, T_Φ, for a given clocking scheme can be represented as the following linear program:

$$\begin{aligned} &\text{minimize} && T_\Phi \\ &\text{subject to} && X_i - X_j \leq E_{p(i),p(j)} - d_{ij} \quad i \hookrightarrow j \mid i,j \in \Psi \end{aligned} \tag{8}$$

Solving the above linear program we obtain the minimum clock period and the GDT's corresponding to it. Our strategy is to transform the GDT solution to a retiming solution to achieve the minimum clock period.

For a given circuit, d_{ij} is constant and for a given clock schedule that has a fixed T_Φ, $E_{p(i),p(j)}$ is constant. Therefore, the constraint matrix reduces to a system of difference constraints. A feasible solution to the constraint matrix exists if the corresponding constraint graph contains no positive cycles [23]. Thus we can solve the, linear program by performing a binary search on the clock period T_Φ. The minimum clock period corresponds to the smallest value of T_Φ at which no positive cycle exists.

The constraint graph has a vertex for each latch in the circuit and one for the host node representing the primary inputs and outputs. There is an edge (i, j) from vertex i to vertex j if there is a purely combinational path from latch i to latch j. The weight on this edge is a function of the clock period T_Φ and is given by $d_{ij} - E_{p(i),p(j)}$. The Bellman Ford algorithm [23] is applied as in [15] using the same speedup techniques which provide a fast implementation. The GDT's at all primary inputs and primary outputs are assumed to be zero. The values of d_{ij}'s are obtained efficiently by using the method in [24].

Notice that number of variables in this constraint graph is equal to the number of latches $|\Psi|$ in the circuit, and the constraints are only between latches with a purely combinational path between them. Therefore this constraint graph in practice, is much smaller and sparse as compared to the traditional constraint graphs of [13, 14], which have one variable for every gate and constraints to all reachable gates. Further unlike the traditional methods of [13, 14], which need to construct the larger and denser constraint graph for every binary search point (by solving an all-pair shortest path problem), our constraint graph needs to be constructed only once. At each point in the binary search the constraint graph can be obtained by a simple reweighting of the graph edges. Therefore the run time of this binary search is much less than that of the traditional methods.

This optimal clock period with skews is called P_s, and it is same as the maximum delay-to-register ratio of [25]. Both are lower bounds on clock period obtainable via retiming. However, instead of using it just as a lower bound (as in [25]), we use it to obtain the amount by which each latch is required to move in order to satisfy the clock period. This amount is then used to obtain a retiming solution as described next.

4.2 Phase B: GDT Minimization

In Phase B, we reduce the GDT values obtained in Phase A by applying retiming transformations using Theorem 1. This procedure relocates the latches with nonzero GDT's across logic gates, while maintaining the optimal clock period previously found. Because of the freedom provided to D_i by the active interval of clock phase $p(i)$ (which allows D_i to be set to any value between $T_\Phi - T_{\phi_{p(i)}}$ and T_Φ), $S_i = 0$ can be achieved if $-T_{\phi_{p(i)}} \leq X_i \leq 0$. If S_i cannot be set to zero, we try to bring X_i as close to 0 or $-T_{\phi_{p(i)}}$ as possible so as to minimize the magnitude of the final skew S_i.

Although the method for FF relocation presented in [15] can be modified to work for latches, we present an equivalent yet conceptually simpler method of GDT minimization by latch relocation. A gate can be retimed in forward [backward] direction if it has latches on all of its inputs [outputs], this retiming will result in removing one latch from all its inputs [outputs] and adding one latch to all its outputs [inputs].

We maintain two sets one for the gates that are to be forward retimed and one for the gates that are to be backward retimed. The forward retiming set F is initialized to contain all gates that have at least one latch on all their inputs. Similarly the backward retiming set B is initialized with gates that have at least one latch on all their outputs. We then process these sets by taking a gate from the set and retiming it, if the skew on the latches can be reduced by this retiming. After every retiming the sets are updated. The pseudo code for this is listed as the function `retime()` in Figure 10.

```
retime()
{
  F = {v|v ∈ V and w(e_uv) ≥ 1 ∀u ∈ FI(v)} /* initialize F */
  B = {v|v ∈ V and w(e_vu) ≥ 1 ∀u ∈ FO(v)} /* initialize B */
  while(∃ u ∈ F) do forward_retime(u, F);
  while(∃ u ∈ B) do backward_retime(u, B);
}
```

Figure 10: Function retime().

The functions `forward_retime(gate,set)` and `backward_retime(gate,set)` retime the gate if needed, and update the respective sets. The corresponding pseudocodes are listed in Figure 11.

The function `GDT_to_skew(GDT)` converts a GDT value to the corresponding minimum magnitude skew using relation (7). For forward retiming of a gate v the effective GDT before retiming, X_i is given by the maximum GDT at its inputs, while the effective GDT after retiming X_i' is given by $X_i' = X_i + d(v)$. For backward retiming the effective GDT before retiming X_i is given by the minimum GDT at its outputs, and the GDT after retiming X_i' is given by $X_i' = X_i - d(u)$. In either case the gate is retimed only if the magnitude of the effective skew after retiming S_i' is less than the magnitude of the effective skew before retiming S_i. As mentioned earlier a gate v is forward [backward] retimed by removing a latch from each of its inputs [outputs] and adding a latch with

```
forward_retime(v, F)
{
  F ← F − v; /* remove gate v from F */
  X_i = maximum GDT at the inputs of gate v;
  X'_i = X_i + d(v);
  S_i = GDT_to_skew(X_i);
  S'_i = GDT_to_skew(X'_i);
  if (|S'_i| < |S_i|) do
  { /* retime gate v */
    for ∀u ∈ FI(v) do {w(e_uv) ← w(e_uv) − 1}; /* delete a latch from each input */
    for ∀u ∈ FO(v) do {w(e_vu) ← w(e_vu) + 1}; /* add a latch with GDT = X'_i on all outputs */
    for ∀u ∈ FO(v) do
        if (w(e_wu) ≥ 1 ∀w ∈ FI(u)) do F ← F ∪ u; /* update F */
  }
}

backward_retime(v, B)
{
  B ← B − v; /* remove gate v from B */
  X_i = minimum GDT at the outputs of gate v;
  X'_i = X_i − d(v);
  S_i = GDT_to_skew(X_i);
  S'_i = GDT_to_skew(X'_i);
  if (|S'_i| < |S_i|) do
  { /* retime gate v */
    for ∀u ∈ FO(v) do {w(e_vu) ← w(e_vu) − 1}; /* delete a latch from each output */
    for ∀u ∈ FI(v) do {w(e_uv) ← w(e_uv) + 1}; /* add a latch with GDT = X'_i on all inputs */
    for ∀u ∈ FI(v) do
        if (w(e_uw) ≥ 1 ∀w ∈ FO(u)) do B ← B ∪ u; /* update B */
  }
}
```

Figure 11: Functions forward_retime() and backward_retime().

GDT X_i' to all its outputs [inputs]. If after forward [backward] retiming a gate v, any of its fanout [fanin] gate w now has at least one latch on all its fanins [fanouts], then we add gate w to the forward [backward] set F [B].

Retiming a latch forward across a gate u affects the edge weights on only its own fanouts and not the edge weights on fanouts of any other gate. Therefore forward retiming a gate u cannot enable the backward retiming of any other gate that could not be previously retimed in the backward direction. Since we forward retime a gate u only if the effective skew magnitude reduces by this retiming, and not if it remains the same, a gate u cannot be backward retimed after it has been forward retimed once (even though it may have latches on all its fanouts), because this backward retiming will increase the skew magnitude. Therefore a gate can never be retimed in both the forward and backward direction. Thus forward retimings have no effect on backward retimings and both types of retimings can be carried out independently. Due to this reason we do have to process the forward set again after it has been processed once.

4.3 Retiming for a Target Period

Retiming a circuit for a given target clock period is a special case of the minperiod retiming problem. In this problem we are given a circuit and a clock schedule Φ that has a fixed T_Φ. If the given clock schedule is feasible, the method should return a retimed circuit that is properly clocked. If the clock schedule is not feasible the method should indicate so. For this problem we do not need to perform the binary search in Phase A. The constraint graph is constructed as earlier and the Bellman-Ford algorithm is applied on it to obtain the set of required GDT's. If the Bellman-Ford algorithm detects a positive cycle the clock scheme is not feasible, and is reported as such, otherwise Phase B is performed.

Due to the flexibility in the non-critical part of the circuit, and the transparent nature of the latches, retiming for a given clock period is not unique, and different retimed circuits can be obtained all of which satisfy the target clock period. As an example for the circuit in Figure 1 two different retimings are shown in Figure 2 and Figure 3 for the same target clock period of 2.0 units. Our objective in minperiod retiming is to find one of these possible solutions efficiently, with as few retiming moves as possible. As in [15] we initialize the GDT's to 0 in the Bellman-Ford algorithm, and take advantage of the slacks to minimize the number of moves. For minperiod retiming of the circuit in Figure 1 our method will generate the circuit in Figure 2, since it requires fewer latch motions than the circuit in Figure 3.

4.3.1 The ALAP and ASAP Retimings

Out of the set of all possible retimings for the given clock scheme, two are of particular interest. We can obtain a retiming such that all latches move as far as possible to the left. This is called "as soon as possible (ASAP)" retiming. Similarly the retiming that moves all the latches as far as possible to the right is referred to as the "as late as possible (ALAP)" retiming. Both ASAP and ALAP retiming assume no latch is moved across the host node (H). These ASAP and the ALAP locations

can be seen as the extreme locations of latches in the circuit for the given clock scheme, and will be utilized, as in [6], in Section 5 for efficient minarea retiming. For the circuit in Figure 1 the ALAP and ASAP retimings are shown in Figure 2 and Figure 3 respectively. As in [6] these ASAP and ALAP retimings can be obtained by modifying the minperiod retiming algorithm.

Unlike retiming for a given period, in ALAP retiming our objective is to move the latches to the right, as much as possible. For this we initialize all GDT's to $-\infty$, before applying the longest path Bellman-Ford algorithm to the constraint graph. In Phase B we use the allowable range of GDT's to move a latch to the right as much as possible, i.e., if the new GDT after moving a latch to the right is still within the allowable range, we move the latch to the right. Notice that this is done even if the original GDT was within the allowable range.

In ASAP retiming we obtain the GDT's by running the Bellman-Ford algorithm on the transpose [23] of the original constraint graph (i.e., a graph with the same vertex set as the original graph, but with the edge directions reversed) with all latches initialized to $-\infty$. Since all the edge directions where reversed the longest path values for all latches must undergo a sign reversal to obtain the correct GDT values.

4.4 A Bound on the Clock Period of the Retimed Circuit

Theorem 2 *At the end of the retiming procedure in Phase B, the magnitude of skew at each latch i, is no more than*

$$\gamma_{p(i)} = \max\left(0, \frac{M - T_{\phi_{p(i)}}}{2}\right) \tag{9}$$

where M is the maximum delay of any gate in the circuit.

Proof: We have two cases

Case A : $\underline{M \leq T_{\phi_{p(i)}}}$ If the maximum gate delay is less than the active duration of the clock, we need to prove that at the end of Phase B, all latches will have zero skew. We will prove this by contradiction, assume that a latch i has nonzero skew S_i at the end of Phase B. We have two subcases.

Case 1: $\underline{S_i > 0}$ In this case the GDT of latch i is $X_i = S_i$. The new GDT of the latch after it is moved left across a gate of delay d_1 is given by $X_i' = X_i - d_1$. Since $d_1 \leq M \leq T_{\phi_{p(i)}}$ we have $X_i' \geq -T_{\phi_{p(i)}}$, thus the effective skew S_i' after this possible move is either zero (if $T_{\phi_{p(i)}} \leq X_i' \leq 0$), or $|S_i'| < |S_i|$. In either case the latch i can be moved left across the gate and have its skew reduced. This contradicts the assumption that Phase B is complete.

Case 2: $\underline{S_i < 0}$ If $S_i < 0$ then the GDT of latch i is negative, i.e., $X_i = S_i - T_{\phi_{p(i)}}$, and the proof is similar to case 1.

Case B: $\underline{M > T_{\phi_{p(i)}}}$ If the maximum gate delay is more than the active duration of the clock, we need to prove that for any latch i, at the end of Phase B the skew magnitude is less

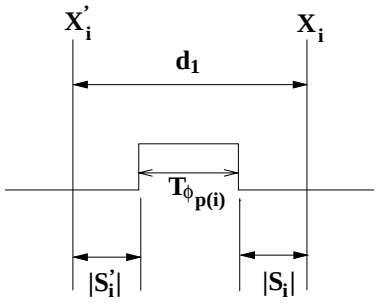

Figure 12: Worst-case situation for remaining skew.

than $\gamma_{p(i)}$. Phase B is complete only when for every latch i we have $|S_i'| \geq |S_i|$, where S_i is the current skew and S_i' is the skew after a possible move across a gate with delay d_1. As shown in Figure 12 the largest possible final skew magnitude corresponds to the situation when $|S_i| = |S_i'|$. In this case we have $d_1 = 2 \cdot |S_i| + T_{\phi_{p(i)}}$ and hence $|S_i| \leq \frac{d_1 - T_{\phi_{p(i)}}}{2}$. Since M is the maximum gate delay this implies that $|S_i| \leq \frac{M - T_{\phi_{p(i)}}}{2}$.

Theorem 3 *If, in a k phase circuit at the end of the retiming procedure all skews are set to zero, then the final clock period (P_r) satisfies the following condition*

$$P_r \leq P_s + \sum_{i=1}^{k} \max(0, M - T_{\phi_i})$$

where P_s is the optimal clock period with skews found in Phase A, and M is the maximum delay of any gate in the circuit.

Proof : Each difference constraint for the optimal clock period (with skews) is of the form

$$X_i - X_j \leq E_{p(i),p(j)} - d_{ij}.$$

Theorem 2 guarantees us that at the end of Phase B $|X_i|$ and $|X_j|$ are within $\gamma_{p(i)}$ and $\gamma_{p(j)}$ of their optimal values respectively. Therefore the actual value of $X_i - X_j$ after Phase B must lie within $(\gamma_{p(i)} + \gamma_{p(j)})$ of the required value of $X_i - X_j$ in Phase A. This implies that the inequality that defines the difference constraint can be maintain by increasing $E_{p(i),p(j)}$ by no more than $(\gamma_{p(i)} + \gamma_{p(j)})$. Since each $E_{i,j}$ increases by no more than $(\gamma_{p(i)} + \gamma_{p(i)})$, the clock period $T_\Phi = \sum_{i=1}^{k} E_{p(i),p(j)}$ will increase by no more than $\sum_{i=1}^{k}(2 \cdot \gamma_i)$ or $\sum_{i=1}^{k} \max(0, M - T_{\phi_i})$.

For level-clocked circuits if M, the maximum delay of any gate in the circuit, is less than the active interval of the clock T_{ϕ_i} for all i, then our method will always achieve the optimal solution. Even if this condition is not true, the bound in Theorem 3 is not expected to be large in practice since it is the difference between gate delay and active interval of clock. Further since it is a worst case bound from a possibly unachievable clock period P_s, we expect the final result to be close to the optimal clock period obtainable by retiming.

The method presented here is also applicable to circuits that are not well-formed, as defined in [14], or contain multi-cycle paths. For such circuits, the only requirement is to compute the time available for data to travel between two latches connected by a purely combinational path. This can be done either using the phase shift operator, or the timing assertions placed by the designer. In phase B, latches are moved across a gate only if they all have the same phase. Such circuits may have short path violations, which may be resolved by using min-padding [18].

5 Minimum Area Retiming

Although the minperiod retiming algorithms can achieve significant improvement in the clock period, they pay no regard to the number of latches in the circuit. As a result minperiod retiming can significantly increase the number of latches in the circuit, and hence the circuit area and power. To contain this increase we perform constrained minarea retiming. Performing a constrained minarea retiming with the target period set to the period obtained by minperiod retiming, will give us the fastest circuit with least area overhead.

The minarea retiming problem can be modeled as a LP [2]. Unfortunately under general clock schedules with unequal phases the minarea retiming problem must be modeled as a general integer linear program of the type given in [26], while restricting the clock scheme to a symmetric multi-phase clock enables us to model the minarea retiming problem as an efficiently solvable LP (dual of min-cost flow problem) [13]. Therefore in this work we will consider only symmetric clock schemes. As the LP presented in [13] has almost $\frac{|G|^2}{2}$ constraints for a circuit with $|G|$ gates, minarea retiming of large circuits is not feasible. In this section we present an efficient method for minarea retiming of large level-clocked circuits, without sacrificing optimality. Our approach is to improve the efficiency of minarea retiming by
(a) reducing the size of the LP,
(b) generating this LP faster, and
(c) solving the LP efficiently.
Reducing the size of the LP reduces the space requirement of minarea retiming making it feasible for large circuits. Efficient techniques for generating the LP are essential to retime large circuits in reasonable times. Lastly since the size of even the reduced LP will be significant, efficient algorithms for solving it are imperative. We use the efficient network simplex method from [16] for this purpose. Using this technique we could solve a problem with 70,000 variables and 8.2 million constraints in about 9 minutes [6].

In this section we first present the LP formulation of minarea retiming. We then reduce the size of this LP, both in terms of number of variables and constraints, without sacrificing any optimality. Finally we present efficient techniques for generating this LP.

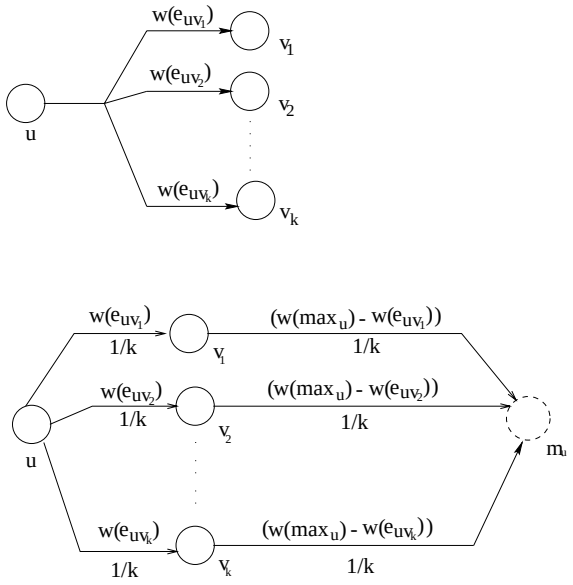

Figure 13: Maximal Latch Sharing

5.1 The Minarea Linear Program

The minarea retiming LP for level-clocked circuits is similar to the LP for edge-triggered circuits given in [2]. The decision variables of this LP are the r variables of the gates, and the objective function represents the number of latches added to the retimed circuit in relation to the original circuit. Since the latches at the output of a gate can be combined or shared, we take into account maximal latch sharing to accurately model the number of latches. To achieve this, we associate a mirror vertex m_u of zero delay with every multiple-fanout gate u, as shown in Figure 13 [2]. All edges from the gate u to its fanouts, and the edges from these fanouts to the mirror vertex m_u have a width of $1/k$, where k is the number of fanouts. The edge weights are as shown in the figure, where $w(max_u) = \max_{\forall v \in FO(u)} w(e_{uv})$ is the maximum weight over all edges from the gate u. Further details of this maximal latch sharing model can be found in [27]. Note that due to the introduction of these mirror vertices we now have two nodes for every multiple-fanout gate, thus $|G| \leq |V| \leq 2 \cdot |G|$.

The LP contains two set of constraints. 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 will ensures that after retiming, the circuit is properly clocked by the target clock T_Φ, and is referred to as the *period constraint set* C_p. We will now derive these period constraints.

For a k-phase symmetric clock we have $T_{\phi_i} = T_\phi \ \forall i = 1 \cdots k$ and $\pi = \frac{T_\Phi}{k}$. For a level-clocked circuit to be properly clocked the total delay on any path should be less than the time available. The time available for a path p with $w_r(p)$ latches is $(w_r(p) + 1) \cdot \pi$, in addition cycle stealing can provide up to T_ϕ additional time. Thus as in [25] we get the following condition for proper clocking:

$$d(p) \leq \quad (w_r(p) + 1) \cdot \pi + T_\phi \quad \forall\, p : u \rightsquigarrow v \tag{10}$$

This constraint can be rewritten after substitution of Equation 1 as

$$r(u) - r(v) \quad \leq w(p) - \frac{d(p)}{\pi} + 1 + \frac{T_\phi}{\pi} \quad \forall\, p : u \rightsquigarrow v \tag{11}$$

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.,

$$\delta(u, v) = \min_{\forall p : u ; v} \left(w(p) - \frac{d(p)}{\pi} \right) \tag{12}$$

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

$$\Delta(u, v) = \left\lfloor \delta(u, v) + \frac{T_\phi}{\pi} + 1 \right\rfloor \tag{13}$$

Since the retiming variables $r(u)$ and $r(v)$ are integers, we can rewrite relation (11) as the period constraints

$$r(u) - r(v) \quad \leq \Delta(u, v) \quad \forall\, p : u \rightsquigarrow v \tag{14}$$

We now have the constrained minarea retiming LP as:

$$\begin{array}{lll} \text{minimize} & \sum_{v \in V} \left[\left(\sum_{\forall j \in FI(v)} \beta(e_{jv}) - \sum_{\forall j \in FO(v)} \beta(e_{vj}) \right) \cdot r(v) \right] & \\ \text{subject to} & r(u) - r(v) \leq w(e_{uv}) & \forall e_{uv} \in E \\ & r(u) - r(v) \leq \Delta(u, v) & \forall u, v \;\; \exists \;\; p : u \rightsquigarrow v \end{array} \tag{15}$$

Notice that all the constraints in this LP are difference constraints of the form $r(i) - r(j) \leq c_{ij}$ where $c_{ij} = w(e_{uv}) \;\; \forall e_{uv} \in E$ and $c_{ij} = \Delta(u, v) \;\; \forall u, v \;\; \exists \;\; p : u \rightsquigarrow v$.

5.2 Reducing the Linear Program

To reduce the space requirements of minarea LP it is imperative that we have some techniques to prune the constraints as they are generated, rather than after all the constraints have been generated. Since the required effort in solving a LP is strongly dependent on the number of variables, and the number of variables in Equation (15) can be up to $2 \cdot |G|$, it would help to reduce the number of variables as well. In this section we will take advantage of the relationship between retiming and GDT presented in Section 3 to reduce the size of the LP by using bounds on the r variables [6].

As in [6], the ALAP and ASAP retimings described in Section 4.3.1 give us bounds on the r variables, of the form

$$L_u \leq r(u) \leq U_u \tag{16}$$

We refer to L_u as the lower bound and U_u as the upper bound on gate u. Like the ASAP and ALAP retimings, these bounds are with reference to a fixed host vertex, i.e., $L_H = U_H = 0$. If $U_u = L_u = k_u$ we say that gate u is fixed or immobile. On the other hand if $U_u \neq L_u$ we say that gate u is flexible

or mobile. These bounds give us a reduced variable set $V' \subseteq V$ as

$$V' = \{v \in V | U_v \neq L_v\} \tag{17}$$

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

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

The presence of the bounds obtained in inequality (16) makes a large number of constraints redundant, i.e., these constraints are implied by the bounds. We now present a rule to identify these redundant constraints.

Rule 4 *Any constraint (i, j) of the form $r(i) - r(j) \leq c_{ij}$ is redundant in the presence of the bounds of inequality (16) 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 inequality (16) 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 $U_i - L_j \leq c_{ij}$.

To obtain the reduced circuit constraint set $C_c' \subseteq C_c$ we accept only those constraints from C_c that are not dropped by the application of Rule 4. Thus

$$C_c' = \{(i, j) \in C_c | U_i - L_j > w(e_{ij})\} \tag{18}$$

The reduced period constraint set $C_p' \subseteq C_p$ is similarly obtained by applying Rule 4, and some other techniques for pruning redundant constraints presented later in Section 5.3.3. Using these techniques we can reduce the LP in inequality (15) to a much smaller LP given below

$$\begin{array}{lll} \text{minimize} & \sum_{v \in V'} \left[\left(\sum_{\forall j \in FI(v)} \beta(e_{jv}) - \sum_{\forall j \in FO(v)} \beta(e_{vj}) \right) \cdot r(v) \right] + K & \\ \text{subject to} & r(u) - r(v) \leq w(e_{uv}) & \forall (u, v) \in C_c' \\ & r(u) - r(v) \leq \Delta(u, v) & \forall (u, v) \in C_p' \\ & L_u \leq r(u) \leq U_u & \forall u \in V' \end{array} \tag{19}$$

where K is a constant accounting for the increase in the number of latches due to fixed gates, and is added to ensure that this LP has the same optimal function value as the LP in Equation (15). Its value is given by

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

Bounds on the mirror vertices introduced to model the maximal latch sharing can be obtained directly from the bounds on fanout gates as in [16] and are given by Theorem 5. The constraints associated with these mirror vertices can also be obtained by direct inspection of the circuit. Therefore we do not need to explicitly add these mirror vertices to the circuit graph. Since every multi-fanout gate has a mirror vertex, this gives us important savings in terms of the space and time requirements.

Theorem 5 *The bounds on the r value of a mirror vertex m_i of gate i in Figure 13 can easily be derived from the bounds on the fanout gates and are given by*

$$\begin{aligned} U_{m_i} &= \max_{\forall j \in FO(i)} (U_j + w(e_{ij})) - w(max_i) \\ L_{m_i} &= \max_{\forall j \in FO(i)} (L_j + w(e_{ij})) - w(max_i) \end{aligned} \tag{20}$$

5.3 Generating the Reduced Linear Program

A major portion of the effort in retiming a level-clocked circuit for minimum area is spent in generating the period constraints set C_p. We now describe efficient techniques for generating this set C_p'. The generation of period constraints requires computation of δ values for all-pairs of gates in the circuit. However if the ALAP retiming satisfies the target clock period[2], then we need to compute δ values only from flexible gates, as stated in the following theorem.

Theorem 6 *If the ALAP retiming satisfies the target clock period, any period constraint from a fixed node a (i.e. $U_a = L_a$) is redundant in the presence of the bounds of Equation (16) and need not be generated.*

Proof : Since ALAP positions are feasible solutions the following holds for all constraints in C_p.

$$L_i - L_j \leq \Delta(i,j) \;\; \forall \, (i,j) \in C_p \tag{21}$$

[2]Notice that if the maximum gate delay in the circuit is more than the active period T_ϕ, it is possible for ALAP retiming to violate the target clock period even if the target clock period is feasible by retiming alone. This is because the method of finding ALAP retiming converts a (continuous) skew optimization solution to a (discrete) retiming solution. This, however, does not imply that these ALAP bounds are wrong, but merely that they are not tight enough. In level clocked circuits, due to the flexibility offered by the transparent nature of latches it is very unlikely that the ALAP retiming will violate the target clock period. In our experiments, we used randomly-assigned delays between 1 and 20 units ($M = 20$) and a 50% duty cycle, and found that no ALAP retiming violated the target clock period. We did not see any significant effect of M on the run time of our method.

Consider any period constraint $(a, b) \in C_p$ from a fixed gate a, to any other gate b of the form $r(a) - r(b) \leq \Delta(a, b)$. By relation (21) $L_a - L_b \leq \Delta(a, b)$, and by Equation (16) $r(a) - r(b) \leq U_a - L_b$. Because gate a is fixed $U_a = L_a$, therefore $r(a) - r(b) \leq L_a - L_b \leq \Delta(a, b)$. Thus the constraint (a, b) is redundant and can be dropped. Since this is true for any period constraint from gate a, we do not need to generate any period constraint from a fixed gate, as they will all be redundant.

5.3.1 Computing the δ Values

The δ values can be obtained by re-weighting each edge e_{ij} with $w'(e_{ij}) = \left[w(e_{ij}) - \frac{d(i)}{\pi}\right]$ and computing all-pair shortest paths. Direct computation of all pair shortest paths, e.g., using Floyd-Warshall algorithm [23] requires $O(|V|^2)$ memory and is not practical for large circuits with tens of thousand gates. Therefore we solve multiple single-source shortest path problems, once for every gate as a source, and use Johnson's algorithm [23] for this purpose. Johnson's algorithm first re-weights all edges to ensure nonnegative edge weights. The shortest paths are then computed by running Dijkstra's algorithm for each gate as source.

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, reachable from gate a.

$$\begin{aligned} \text{By definition,} \quad r(a) - r(b) &\leq U_a - L_b \\ \text{If} \quad U_a - L_b &\leq \delta(a, b), \\ \text{then} \quad r(a) - r(b) &\leq \delta(a, b). \end{aligned} \tag{22}$$

From relation (11) and (12)

$$r(b) - r(c) \leq \delta(b, c) + \frac{T_\phi}{\pi} + 1,$$

which when combined with relation (22) gives

$$r(a) - r(c) \leq \delta(a, b) + \delta(b, c) + \frac{T_\phi}{\pi} + 1 \tag{23}$$

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 do not need to process the fanouts of gate b to obtain $\delta(a, c)$. On the other hand, if 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 relation (23) is same as the period constraint $r(a) - r(c) \leq \Delta(a, c)$. If $U_a - L_b \leq \delta(a, b)$ then this period constraint is redundant. In either case we need not process the fanouts of gate b. Since this is true for any c, reachable from gate a, and we are interested only in gates reachable from gate a, we get the following rule.

Rule 7 *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.*

The idea of limiting the search depth by the clock period during constraint generation in minarea retiming of edge-triggered circuits was used in [5] and [28]. This idea is not directly applicable to level-clocked circuits, since constraint generation cannot be limited by the clock period due to cycle stealing. Notice that the condition $r(a) - r(b) \leq \delta(a,b)$ is too restrictive and enforcing it will in general lead to suboptimal solutions. However, for the special case where this conditions is implied by the bounds, the relation is true. This can be used to limit the search during constraint generation as shown in Relations (22) and (23). The bounds on the r variables enable the identification of this condition and hence are vital for application of this rule.

We take advantage of the bounds on r variables to speed up the computations, by applying Theorem 6 to compute δ values only from the flexible gates, and using Rule 7 to reduce the computation for the δ values actually computed. We found that this significantly improves the time taken to generate the period constraints.

5.3.2 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. The idea is motivated by the fact that in most practical circuits a high percentage of gates are single-fanout gates. Consider one such single-fanout gate a with fanout b. For the gate a, the shortest paths to all other gates must be via gate b, which implies that $\delta(a,c) = w'(e_{ab}) + \delta(b,c)$. Therefore we can obtain the shortest paths from gate a by simply adding $w'(e_{a,b})$ to the shortest paths from gate b. Thus if we somehow ensure that shortest paths from gate b are obtained just before those from gate a, we will save one complete execution of Dijkstra's algorithm for gate a as source. We call this approach "chaining" and the set of gates for which only one set of δ computations is performed as "chains". We now define a simple chaining technique that stores δ values from only one source, hence we call it "1-chaining".

For 1-chaining a graph $G(V,E)$ we preprocess it to get a set of chains $\Omega = \{\omega^1, \omega^2 \ldots \omega^{|\Omega|}\}$ such that every vertex in the graph is included in exactly one chain, i.e.

$$\omega^i \cap \omega^j = \emptyset \;\; \forall i \neq j \text{ and } \omega^1 \cup \omega^2 \cup \ldots \omega^{|\Omega|} = V$$

Each chain ω^i itself is an ordered list (of size $|\omega^i|$) of vertices in the graph, i.e., $\omega^i =< \omega^i_1, \omega^i_2 \ldots \omega^i_{|\omega^i|} >$. Thus ω^i_j is the j^{th} gate in the i^{th} chain. The first gate ω^i_1 in a chain ω^i is called its head, and all gates in a chain except the head must have only one fanout, i.e., $|FO(\omega^i_j)| = 1 \; \forall i$ and $\forall \; j > 1$. The gates in a chain are ordered such that a gate is the fanout of any gate appearing after it in the chain, i.e.,

$$e_{\omega^i_{j+1}\omega^i_j} \in E \;\; \forall i \text{ and } 1 \leq j < |\omega^i|$$

We only need to obtain the δ values from the gates that are at the head of a chain, i.e., we only need to compute the values $\delta(\omega^i_1, u) \; \forall u \in V$ and $1 \leq i \leq |\Omega|$. For all other gates the δ values can be

obtained by adding the re-weighted edge weight to the δ values from its fanouts, i.e.,

$$\delta(\omega^i_{j+1}, u) = \quad \delta(\omega^i_j, u) + w'(e_{\omega^i_{j+1}\omega^i_j}) \quad \forall u \in V \text{ and } 1 \leq j < |\omega^i| \text{ and } 1 \leq i \leq |\Omega|$$

Notice that for a gate that is not at the head of any chain we obtain the δ values by a simple addition, instead of a full run of Dijkstra's algorithm. Since we need to run Dijkstra's algorithm only for gates at the head of a chain we need to perform only $|\Omega|$ single-source shortest path computations ($|\Omega| \leq |V|$). Thus our goal in obtaining these chain is to reduce there number, i.e., minimize $|\Omega|$. In the worst case where every gate in the circuit has more than one fanout, each chain contains only one gate, and $|\Omega| = |V|$, then we need to perform the complete Johnson's algorithm. 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 [23] with $O(|V|^2)$ memory requirement.

The 1-chaining described above is an intermediate method in which we save 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 $O(k \cdot |V|)$ memory and careful selection of the k sources, and is not considered in this work.

We now describe a simple preprocessing technique to obtain 1-chaining. This preprocessing step first assigns a label to each gate which indicates the number of gates that can reuse its δ computation. All the gates have their labels initialized to 0. These labels are updated by a relaxation step, in which every single-fanout gate relaxes the label of its fanout gate by increasing it (to its own label plus one). Since multiple-fanout gates can not reuse δ computations of their fanout gates in 1-chaining they do not relax the labels of their fanout gates. This relaxation process is finite because we cannot have cycles containing only single-fanout gates. The chains are then formed by initializing a queue with all multiple-fanout gates. Every gate in this queue starts a new chain. For the gate at the end of the chain, we process the fanin gates, adding the single-fanout gate with the highest label (amongst the fanins) to the chain; all other gates in the fanin are added to the queue. The fanins of the gate now at the end of the chain are processed similarly, until no more gates can be added to this chain. This procedure is repeated until the queue is empty.

We found that, on an average we could reduce the time spent in generating the period constraints by about 50% using the simple 1-chaining technique described above. The time spent in preprocessing to obtain 1-chaining is very small, making it a useful procedure even if only a small number of gates have single fanout. As a side note, Rule 7 must be modified for use with chaining to ensure it holds for all gates that reuse the δ computation.

5.3.3 Additional Constraint Pruning Techniques

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.

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

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

If $\Delta(a,b) + w(e_{bc}) \leq \Delta(a,c)$ then constraint $C3$ is redundant and can be dropped. This leads us to the following rule

Rule 8 *If b and c are two gates reachable from gate a, such that 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.*

Since we generate the period constraints from one gate (say gate a) at a time, both $\Delta(a,b)$ and $\Delta(a,c)$ are available in the same iteration. Further because gate b is a fanin of gate c the value $w(e_{bc})$ is available directly from the circuit graph. Therefore Rule 8 can be efficiently applied to drop redundant period constraints as they are generated. This reduces the space (memory) requirement of the period constraints.

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

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

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

To apply Rule 9 we require the value of $\Delta(b,c)$ and $\Delta(a,c)$. Since we generate period constraints from one gate at a time, the period constraints to a gate (c) from two different sources (gate a and b) cannot be efficiently accessed. Thus it would appear that Rule 9 cannot be efficiently applied. However because of the reuse of δ computation described in Section 5.3.2, Rule 9 can be efficiently applied if gate a has only one fanout (gate b). This is possible because $\Delta(a,c)$ is derived from $\Delta(b,c)$, and hence both are available when the period constraint from a to c is being generated. Thus we can drop redundant period constraints from gate a as they are generated.

Rule 4 is valid only in presence of the bounds and it prunes the constraint sets because the information in these bounds make some constraints redundant. Rule 8 and Rule 9 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. Rule 8 and Rule 9, can be generalized to include implication by more than two constraints; these generalized rules will, however, be computationally expensive to apply.

6 Experimental Results

Table 1: Quality of Retiming for Single Phase Circuits

Circuit	$\|G\|$	Period			# Latches			CPU time	
		P_i	P_s	P_r	$\|\Psi_i\|$	$\|\Psi_p\|$	$\|\Psi_a\|$	T_{period}	T_{area}
s3384	1,685	84.0	38.5	38.5	183	326	164	0.23s	2.34s
s4863	2,342	116.0	59.0	59.0	104	254	114	0.22s	4.94s
s5378	2,779	48.0	48.0	48.0	179	263	143	0.21s	2.99s
s6669	3,080	118.7	49.0	49.0	239	472	278	0.56s	4.09s
s13207.1	7,791	127.0	120.0	120.0	627	890	446	1.09s	13.94s
s15850.1	9,617	187.0	147.0	147.0	527	869	533	1.84s	38.26s
s35932	16,065	77.0	71.0	71.0	1728	2076	1795	2.81s	63.21s
s38584.1	19,253	125.0	118.0	118.0	1426	3298	1427	4.10s	1m:49.76s
s38417	21,370	68.7	56.0	56.0	1564	2436	1360	4.20s	5m:28.60s
myex1	28,946	256.0	216.0	216.0	1953	4332	1958	8.75s	5m:32.08s
myex2	40,661	104.0	97.0	97.0	2990	6197	2763	9.28s	23m:14.83s
myex3	56,751	137.0	119.0	119.0	4718	8918	4533	14.24s	1h:02m:22.48s

Table 2: Quality of Retiming for Two Phase Circuits

Circuit	$\|G\|$	Period			# Latches			CPU time	
		P_i	P_s	P_r	$\|\Psi_i\|$	$\|\Psi_p\|$	$\|\Psi_a\|$	T_{period}	T_{area}
s3384	1,685	126.0	38.5	38.5	366	638	337	0.40s	2.56s
s4863	2,342	117.0	59.0	59.0	208	473	234	0.29s	5.36s
s5378	2,779	48.0	48.0	48.0	358	480	286	0.29s	3.22s
s6669	3,080	178.0	49.0	49.0	478	960	542	0.76s	6.17s
s13207.1	7,791	127.0	120.0	120.0	1,254	1,795	890	1.48s	18.61s
s15850.1	9,617	187.0	147.0	147.0	1,054	1,777	1,041	2.53s	45.82s
s35932	16,065	77.0	71.0	71.0	3,456	4,144	3,523	3.98s	67.26s
s38584.1	19,253	125.0	118.0	118.0	2,852	7,558	2,852	5.02s	1m:57.52s
s38417	21,370	103.0	56.0	56.0	3,128	4,938	2,766	30.45s	6m:26.99s
myex1	28,946	256.0	216.0	216.0	3,906	9,065	3,891	10.08s	6m:37.48s
myex2	40,661	128.0	97.0	97.0	5,980	13,820	5,551	11.25s	31m:16.52s
myex3	56,751	137.0	119.0	119.0	9,436	17,019	9,041	17.46s	1h:19m:43.07s

We performed retiming on the complete ISCAS-89 benchmark suite, but present results only on the larger circuits. Due to unavailability of large circuits we combine circuits from the ISCAS-89

benchmark suite to obtain circuits (myex1 through myex3) with up to 56,000 gates. We present results for both minarea and minperiod retiming on single phase and two phase circuits. The presented results are for a duty cycle and phase ratio of 50%. In absence of delay information in the ISCAS-89 circuits we assign random delay values (between 1.0 and 20.0 units) to each gate. As in [13] we convert the edge-triggered circuits in ISCAS-89 benchmark to a k phase level-clocked circuits by replacing each FF by k latches.

Table 3: Reduction in the Size of LP for Single Phase Circuits

Circuit	G_{fx}	F_{avg}	# Variables			# Constraints		
			Minaret-L	Original	$R_{variables}$	Minaret-L	Original	$R_{constraints}$
s3384	9.18%	2.59	1,988	2,166	8.22%	33,103	761,365	95.65%
s4863	17.28%	1.21	2,497	2,995	16.63%	32,880	5,481,911	99.40%
s5378	25.50%	1.09	2,728	3,664	25.55%	17,121	4,595,645	99.63%
s6669	26.38%	0.98	3,089	4,100	24.66%	14,267	1,923,524	99.25%
s13207.1	19.97%	3.00	7,449	9,180	18.86%	45,563	22,908,799	99.80%
s15850.1	23.46%	1.88	8,813	11,332	22.23%	64,283	39,493,334	99.84%
s35932	8.43%	2.66	20,071	21716	7.58%	145,978	130,080,328	99.89%
s38584.1	14.21%	2.20	20,501	23,390	12.35%	118,771	293,482,797	99.96%
s38417	1.51%	4.66	25567	25,923	1.37%	1,289,378	149,492,588	99.14%
myex1	13.27%	2.32	30,287	34,417	12.00%	142,525	504,055,977	99.97%
myex2	4.25%	4.34	47,409	49,214	3.67%	1,608,132	819,701,299	99.80%
myex3	1.36%	5.19	69,546	70,414	1.23%	3,608,210	1,624,913,333	99.78%

Table 4: Reduction in the Size of LP for Two Phase Circuits

Circuit	G_{fx}	F_{avg}	# Variables			# Constraints		
			Minaret	Original	$R_{variables}$	Minaret	Original	$R_{constraints}$
s3384	8.15%	5.22	2,006	2,166	7.39%	55,980	761,365	92.65%
s4863	10.51%	2.30	2,706	2,995	9.65%	72,451	5,481,911	98.68%
s5378	19.32%	2.23	2,970	3,664	18.94%	31,765	4,595,645	99.31%
s6669	10.04%	1.92	3,735	4,100	8.90%	20,841	1,923,524	98.92%
s13207.1	17.57%	6.25	7,656	9,180	16.60%	55,395	22,908,799	99.76%
s15850.1	21.60%	3.81	9,013	11,332	20.46%	69,142	39,493,334	99.83%
s35932	7.27%	5.07	20,264	21,716	6.69%	189,068	130,080,328	99.85%
s38584.1	13.78%	4.39	20,590	23,390	11.97%	127,488	293,482,797	99.96%
s38417	0.87%	9.43	25,735	25,923	0.73%	2,446,798	149,492,588	98.36%
myex1	12.63%	4.70	30,489	34,417	11.41%	154,603	504,055,977	99.97%
myex2	1.52%	8.72	48,560	49,214	1.33%	3,638,182	819,701,299	99.56%
myex3	0.67%	10.41	70,000	70,414	0.59%	8,207,036	1,624,913,333	99.50%

Table 1 and Table 2 present the quality of retiming for single phase and two phase circuits respectively. For each circuit the number of gates $|G|$, the initial clock period P_i, the optimal clock period with skews at end of Phase A, P_s, and the final clock period after retiming, P_r, is presented. In all of our experiments retiming is able to achieve the same clock period as skew optimization. This is possible due to the transparent nature of the latches and underscores the usefulness of retiming

level-clocked circuits.

We also show the area cost in terms of number of latches in the initial circuit $|\Psi_i|$, the circuit after minperiod retiming $|\Psi_p|$, and the circuit after constrained minarea retiming with P_r as the target period $|\Psi_a|$. For almost all circuits minarea retiming reduces the number of latches $|\Psi_a|$ in the circuit by a factor of two to three as compared to minperiod retiming $|\Psi_p|$, even though both retime the circuit for the same clock period P_r. This shows the importance of minarea retiming.

The execution time in seconds on a DEC AXP system 3000/900 workstation, with 256M RAM are shown for both minarea retiming T_{area} and minperiod retiming T_{period}. These CPU times include the time spent in all tasks required for retiming except parsing the input circuit file, and highlight the efficiency of our techniques. The CPU times for minperiod retiming T_{period} shown here are less than those reported in [15] for edge-triggered circuits due to the use of the simpler procedure presented in Section 4.2 for latch relocation in Phase B. The CPU time for minarea timing T_{area} was heavily dominated ($> 90\%$ for large circuits) by the time required to generate the LP, this emphasizes the importance of using efficiency-enhancing techniques for generating the LP, e.g., chaining, Rule 7, and Theorem 6.

Table 3 provides a closer look at the reduction in the size of LP for minarea retiming for single phase circuits, while Table 4 has results for two phase circuits. The size of the LP is shown in terms of the number of variables and the number of constraints. Original represents the traditional LP of Equation (15) used in [13] while Minaret-L represents the reduced LP of Equation (19). $R_{variables}$ and $R_{constraints}$ give the percentage reduction in the number of variables and constraints respectively, due to the pruning techniques presented in this work. Also presented are two metrics on the circuits: G_{fx} the number of gates found to be fixed and F_{avg} the average flexibility, i.e., the average values of $(U_y - L_y)$ over all gates in the circuits. The number of variables include both gate and mirror variables and hence the reduction in variables can be different from G_{fx} which does not include mirror vertices. High G_{fx} and low F_{avg} indicates less mobility or flexibility in the circuit, yielding higher percentage reduction in the number of constraints, and faster minarea retiming. It can be seen that up to three orders of magnitude reduction is obtained in the number of constraints by using Minaret-L, e.g., for one phase circuit myex3 the number of constraints reduce from about 1.6 billion to only 3.6 million. The number of unpruned constraints grow at the rate of $O(|G|^2)$ and our pruning techniques reduce this rate of growth significantly.

Although the bounds on the r variables help significantly in reducing the CPU time for minarea retiming, the time spent in obtaining these bounds is an insignificant fraction (less than half a percent) of the total CPU time for minarea retiming. Amongst single phase and two phase circuits the single phase circuits have less flexibility, and a much smaller LP than two phase circuits.

7 Conclusion

Efficient algorithms for both minperiod and minarea retiming of large level-clocked circuits have been presented. The entire ISCAS-89 benchmark suite could be retimed in minutes. The chief

reason for the efficiency of this minperiod algorithm is that it uses the retiming skew relation to map the problem of retiming level-clocked circuits to the much simpler problem of retiming edge-triggered circuit. This enabled us to greatly speedup the process of performing binary search for the optimal clock period. This is possible because we create a potentially small and sparse constraint graph, only once rather than in each step of the binary search as done by traditional methods [13, 14]. The second phase of minperiod retiming is fast because latches do not have to be moved across a large numbers of gates during retiming.

The minarea retiming algorithm is made practical for large circuits by utilizing the retiming-skew relation, and several other pruning techniques (Rule 4, Rule 8 and Rule 9) to reduce the LP in Equation (15) to a much smaller LP in Equation (19), without sacrificing any optimality. A reduction of two to three orders of magnitudes in the number of constraints is obtained for most circuits. The use of Theorem 6, Rule 7, and chaining, greatly speed up the period constraint generation making the overall algorithm very efficient.

In summary, the contributions of this work, which applies retiming-skew relation for fast minarea and minperiod retiming for level-clocked circuits are the following:

- It shows that in practice, large level-clocked circuits can be retimed in reasonable runtime.
- It handles level sensitive latches like edge triggered FF's, thus avoiding a complicated formulation that is forced to handle critical path propagation over several latches. This also avoids the need of generating the constraint graph for every point in the binary search, which is necessitated by the fact that critical paths change with the clock period [14].
- It provides a conceptually simpler technique than [15] for reducing the GDT's in Phase B of minperiod retiming which can also be applied to edge-triggered circuits.
- It provides efficient techniques for generating and pruning the minarea LP.

Some design methodologies may allow a small amount of skew at the FF's. The algorithms presented in this work can also be used to solve the interesting problem of optimizing edge-triggered circuits which allow some skew (less than a given maximum skew magnitude) at the FF's, thus taking advantage of this allowable skew to yield better optimization.

We note that the concept of retiming-skew relation has been used earlier for efficient retiming of edge-triggered circuits in [15] and [16]. However, the complexity added by the transparent nature of latches makes the extension to level-clocked circuits nontrivial. The contributions of this work is therefore in developing and utilizing the retiming-skew relation for level-clocked circuits, and development of efficient generation and pruning techniques for minarea constraints. Without these new techniques it was not possible to perform minarea retiming on the larger circuits.

We would like to point out that although this work makes it possible to retime large level-clocked circuits, much work needs to be done to enable retiming techniques to handle practical circuit issues like, gated clocks and precharged logic. The work in [29] described methods for applying retiming to circuits with gated clocks and precharged logic. We believe that our method can be modified along the same lines to handle gated clocks and precharged logic, and this is a topic for further research.

Acknowledgment

The authors would like to thank the reviewers for their invaluable comments.

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] K. N. Lalgudi and M. Papaefthymiou, "DELAY: An efficient tool for retiming with realistic delay modeling," in *Proceedings of the ACM/IEEE Design Automation Conference*, pp. 304–309, 1995.

[4] T. Soyata, E. G. Friedman, and J. H. Mulligan, Jr., "Incorporating internconnect, register and clock distribution delays into the retiming process," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 16, pp. 165–120, Jan. 1997.

[5] 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.

[6] N. Maheshwari and S. S. Sapatnekar, "An improved algorithm for minimum-area retiming," in *Proceedings of the ACM/IEEE Design Automation Conference*, pp. 2–7, 1997.

[7] J. Monteiro, S. Devadas, and A. Ghosh, "Retiming sequential circuits for low power," in *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design*, pp. 398–402, 1993.

[8] K. N. Lalgudi and M. Papaefthymiou, "Fixed-phase retiming for low power," in *Proceedings of the International Symposium of Low Power Electronic Devices*, pp. 259–264, 1996.

[9] Y. Higami, S. Kajihara, and K. Kinoshita, "Test sequence compaction by reduced scan shift and retiming," in *Proceedings of the IEEE Asian Test Symposium*, pp. 169–175, 1995.

[10] S. Chakradhar and S. Dey, "Resynthesis and retiming for optimum partial scan," in *Proceedings of the ACM/IEEE Design Automation Conference*, pp. 87–94, 1994.

[11] 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.

[12] B. Lockyear and C. Ebeling, "Optimal retiming of multi-phase level-clocked circuits," in *Advanced Research in VLSI and Parallel Systems: Procedings of the 1992 Brown/MIT Conference*, pp. 265–280, 1992.

[13] 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.

[14] B. Lockyear and C. Ebeling, "Optimal retiming of level-clocked circuits using symmetric clock schedules," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 13, pp. 1097–1109, Sept. 1994.

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

[16] N. Maheshwari and S. S. Sapatnekar, "Efficient retiming of large circuits," *IEEE Transactions on VLSI Systems*, pp. 74–83, Mar. 1998.

[17] K. A. Sakallah, T. N. Mudge, and O. A. Olukotun, "Analysis and design of latch-controlled synchronous digital circuits," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 11, pp. 322–333, Mar. 1992.

[18] 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.

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

[20] T. H. Chao, Y. C. Hsu, and J. M. Ho, "Zero-skew clock net routing," in *Proceedings of the ACM/IEEE Design Automation Conference*, pp. 518–523, 1992.

[21] M. Edahiro, "An efficient zero-skew routing algorithm," in *Proceedings of the ACM/IEEE Design Automation Conference*, pp. 375–380, 1994.

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

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

[24] S. S. Sapatnekar, "Efficient calculation of all-pair input-to-output delays in synchronous sequential circuits," in *Proceedings of the IEEE International Symposium on Circuits and Systems*, pp. IV520–IV523, 1996.

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

[26] B. Lockyear and C. Ebeling, "Optimal retiming of level-clocked circuits using symmetric clock schedules," Tech. Rep. UW-CSE-91-10-01, Department of Computer Science and Engineering, University of Washington, Seattle, 1991.

[27] J. B. Saxe, *Decomposable Searching Problems and Circuit Optimization by Retiming: Two Studies in General Transformations of Computational Structures*. PhD thesis, Carnegie-Mellon University, 1985.

[28] A. van der Werf, J. L. Van Meerbergen, E. H. L. Aarts, W. F. J. Verhaegh, and P. Lippens, "Efficient timing constraint derivation for optimally retiming high speed processing units," in *International Symposium on High Level Synthesis*, pp. 48–53, 1994.

[29] A. T. Ishii, "Retiming gated-clocks and precharged circuit structures," in *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design*, pp. 300–307, 1993.

[30] B. Lockyear and C. Ebeling, "The practical application of retiming to the design of high-performance systems," in *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design*, pp. 288–295, 1993.