EXPLOITING LEVEL SENSITIVE LATCHES FOR WIRE PIPELINING

A Thesis

by

VIKRAM SETH

Submitted to the Office of Graduate Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

December 2004

Major Subject: Computer Engineering

EXPLOITING LEVEL SENSITIVE LATCHES FOR WIRE PIPELINING

A Thesis

by

VIKRAM SETH

Submitted to Texas A&M University
in partial fulfillment of the requirements
for the degree of

MASTER OF SCIENCE

Approved as to style and content by:

Jiang Hu (Chair of Committee)	Weiping Shi (Member)
Duncan M. Walker (Member)	Chanan Singh (Head of Department)

December 2004

Major Subject: Computer Engineering

ABSTRACT

Exploiting Level Sensitive Latches for Wire Pipelining.

(December 2004)

Vikram Seth, B.Tech., Indian Institute of Technology Kanpur India

Chair of Advisory Committee: Dr. Jiang Hu

The present research presents procedures for exploitation of level sensitive latches in wire pipelining. The user gives a Steiner tree, having a signal source and set of destination or sinks, and the location in rectangular plane, capacitive load and required arrival time at each of the destinations. The user also defines a library of non-clocked (buffer) elements and clocked elements (flip-flop and latch), also known as synchronous elements. The first procedure performs concurrent repeater and synchronous element insertion in a bottom-up manner to find the minimum latency that may be achieved between the source and the destinations. The second procedure takes additional input (required latency) for each destination, derived from previous procedure, and finds the repeater and synchronous element assignments for all internal nodes of the Steiner tree, which minimize overall area used. These procedures utilize the latency and area advantages of latch based pipelining over flip-flop based pipelining. The second procedure suggests two methods to tackle the challenges that exist in a latch based design. The deferred delay padding technique is introduced, which removes the short path violations for latches with minimal extra cost.

ACKNOWLEDGEMENTS

I would like to express my gratitude to Dr. Jiang Hu for his continued support and guidance. I would also like to acknowledge the assistance of Dr. Min Zhao (Freescale Semiconductor Inc., Austin, TX) for her valuable suggestions. I would like to express my gratitude to Dr. Pasquale Cocchini for giving clarifications on his original work.

I would like to take this opportunity to thank Dr. Hank Walker and Dr. Weiping Shi, for enhancing my knowledge on VLSI Computer Aided Design (CAD) algorithms, through their academic courses. I also thank the Electrical Engineering Department for providing the technical facilities for conducting this work.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

1 INTRODUCTION

The sustained progress of VLSI technology leads to increasing wire delay, shrinking clock period and growing chip size. Industry data [2] shows that the operating frequency of high-performance Integrated Circuits (ICs) approximately doubles every process generation, and the die-size increases by about 25% per generation. Figure 1 and Figure 2 below [15] depict the scaling trends in current and future process generations.

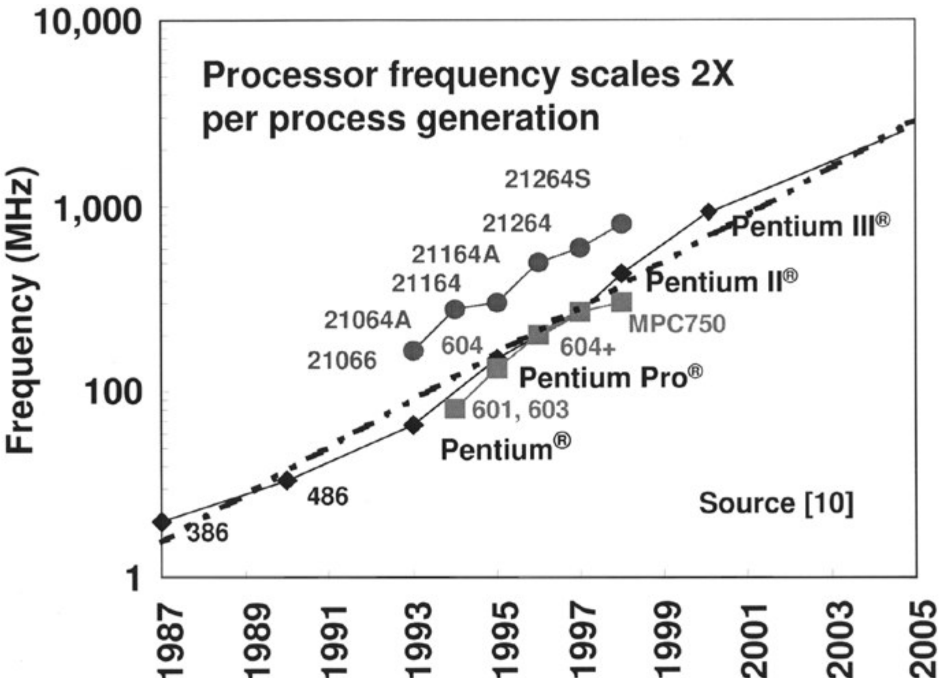

Fig. 1. Frequency scaling trend.

Ideal scaling implies that all dimensions of the wires are shrunk per generation. Therefore, as mentioned in [14], although the wire capacitance per micron doesn't change, the wire resistance per micron doubles every process generation, which results

This thesis follows the style and format of *IEEE Transactions on Computer Aided Design of Integrated Circuits and Systems.*

in wire delay degradation per scaled micron of every generation. The RC-delay of an unbuffered interconnect increases quadratically with wire length. Thus, repeaters have traditionally been used to make the dependence of delay as a linear function of the interconnect length. In an optimally buffered interconnect, the delay of any given stage is approximately equally divided between the repeater and the wire. However, the wire delay degradation in a buffered interconnect, whose dimensions are shrunk every next technology node, during process scaling has led to shrinking of the optimal interval between buffers. Thus, additional repeaters need to be added to optimize the interconnect.

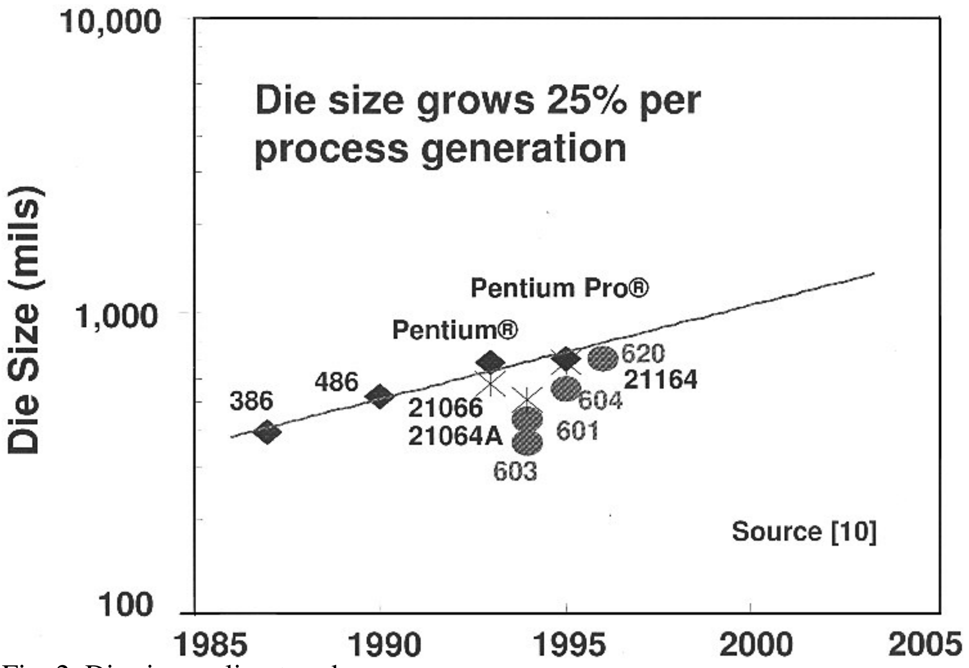

Fig. 2. Die size scaling trend.

The *critical sequential length* [14] is the maximum distance that a signal can travel in an interconnect that has been optimally sized and optimally buffered uniformly, within a single clock period. The work of [14] shows that the critical sequential lengths shrink at the rate of 0.43x per generation. This makes the shrinking not only much faster than normal scaling, it also makes it faster than the rate of decrease (0.57x) in *critical repeater lengths* [14], which is the minimum distance beyond which inserting an optimal-sized repeater makes the interconnect delay smaller than that of the corresponding unrepeated wire. This implies that ideally shrunk interconnects will not only require new repeaters, but also that many of these new repeaters will need to be clocked.

From the above discussion two things become very obvious. Firstly, due to increasing frequency (or decreasing clock period) and die size, the chances of global signals reaching their destination within single clock cycle also reduces. It often takes multiple clock cycles for signals to reach their destinations along global wires. Secondly, the ratio between clocked and unclocked repeaters on a buffered interconnect will continue to grow, as it is scaled across technology nodes. This indirectly points out the fact that the area of these clocked repeaters will become paramount in future technology generations. There have been several works to find solution for routing such multi-cycle global signals across a chip with minimum repeater area.

Traditional interconnect optimization techniques such as repeater insertion [1] deal with single cycle paths only, and are thus inadequate in handling the scenario of synchronous elements insertion (or wire pipelining). The above multi-cycle path routing problem can be solved in two possible ways. The first way, referred to as *wire-pipelining,* is to pipeline the multi-cycle path using synchronous elements, while using buffers to further optimize the interconnect routing. The clock signal is routed to these synchronous elements. In [2], two algorithms are proposed on concurrent repeater and flip-flop insertion for a given Steiner tree. For the case of 2-pin nets, a simultaneous routing, repeater and flip-flop insertion algorithm is suggested in the context of single and multiple clock domains [3]. Given a wire pipelining result, using flip-flops, the work of [4] attempts to improve clock speed through retiming. The second solution is referred to as *wave-pipelining*. In the work of [5], wave pipelining technique, which allows signals to propagate for multiple clock cycles without synchronous elements, is applied for global wires. It eliminates the use of synchronous element along the signal route and allows it to propagate to the destination. This allows the simultaneous existence of several signal wavefronts along a wire path between two synchronous elements. The most important aspect in making this method a success is ensuring that successive waveforms do not interfere. The advantage of using wave-pipelining is that it reduces the clock load and eases clock distribution by reducing the number of synchronous elements in the design. However, wave pipelining is very sensitive to delay, process, and temperature variations – effects that are even more pronounced for long routes. Also, it demands complicated data recovery circuits at the receivers. Thus the latter case of wave pipelining can avoid setup time overhead and intrinsic delay of synchronous elements, but it has its own disadvantages.

The above two methods either adopt edge triggered flip-flops as synchronous elements [2–4] or do not use any synchronizing elements at all [5]. If we look at the spectrum for the degree of synchronization effort during signal propagation, flip-flop insertion is at one extreme of the strongest effort while wave pipelining is at the other extreme of no synchronization effort at all. This work proposes an intermediate level synchronization approach - level sensitive latch based wire pipelining which has no setup time overhead and does not need data recovery circuits. Furthermore, the flexibility in timing constraints of latches allows cycle stealing which may help to reduce both latency [2] and the number of synchronous elements needed, which is the objective in wire pipelining [2]. At the same time, as discussed before, area is a crucial concern because the number of synchronous elements increases by 7× for every process generation and will become a remarkable portion of the total cell count in a chip in near future [2,14]. Since a latch usually is smaller than a flip-flop, it can save both area and power consumption. Therefore, the advantages of using latches in wire pipelining are very important for chip designs in current and next generation technologies.

In circuit designs (vs. interconnect designs), people tend to shun away from using latches because it is more complicated than using flip-flops. One major difficulty is from the cyclic timing constraints caused by feedback loops [6]. However, this difficulty does not exist for wire pipelining due to the absence of feedback loops in interconnect. Moreover, we follow the single phase clock scheme as in many flip-flop based designs so that the multi-phase clock overhead in traditional latch based circuit designs is avoided. Nevertheless, there are two difficulties that need to be overcome for using latches in wire pipelining: (1) input-output timing coupling and (2) the strict short path constraint. This work includes new techniques to solve these difficulties such that the advantages of latches can be fully utilized. In particular, a deferred delay padding technique is developed to correct short path violations with the minimal extra cost. These techniques are integrated with the dynamic programming based framework of [2]. The short path violations are fixed constructively in the dynamic programming procedure instead of as a post processing. Experimental results demonstrate both the advantages of using latches and the effectiveness of our algorithms.

There are two observations that are important for a latch based design. Firstly, the input signal is allowed to arrive/depart anytime within the active clock interval. This puts a restriction on the clock gating technique, which is commonly used in integrated circuits to reduce dynamic power consumption. The clock must not be gated within the active interval, since the propagating signal may be lost. Secondly, since the delay between two latches can be greater than clock period, it is possible that two signal pulses may exist between them. If care is not taken to keep the delay between two buffers less than a clock period, then one pulse may override the other, and result in signal loss.

The text is organized in different sections. Section 2 discusses the advantages and challenges of latch-based designs. Section 3 introduces the 'Concurrent Repeater and Flip-flop Insertion' algorithm [2], which introduces the dynamic programming based framework that is used in this work. Section 4 gives the details of the procedures to use latch insertion to improve latency and area cost. Section 5 gives the experimental results of the algorithms described in previous section. Finally Section 6 discusses the conclusions that can be drawn from this work.

2 USING LATCHES IN INTERCONNECT DESIGN

2.1 ADVANTAGES

The two major objectives of wire pipelining are to reduce latency and area cost. The flexible setup time constraints and smaller size of latches, as compared to flip-flops, can help in achieving the above goals. The area advantage of latches due to smaller size is very evident, as most D-flip-flops are composed of two D-latches. The latency and area advantages of latches due to relieved setup time overhead are described as follows.

The more flexible setup time overhead associated with latches as compared to flip-flops can achieve better latency. This advantage can be illustrated by a first order analysis on a simple case. Consider a two-pin wire with its source at node *u* and its sink at node *v*. If this wire is optimally buffered, the source-sink delay $t_{u,v}$ is asymptotically proportional to its wire length [7]. In flip-flop based approaches, signals depart from one flip-flop at a switching edge of the clock signal and have to arrive at next flip-flop before T - T_{setup} -T_{skew}, where T is clock cycle time, T_{setup} is the setup time and T_{skew} is clock skew between the two flip-flops. If T_{prop} is the propagation delay of a flip-flop, the maximal interconnect delay allowed between two flip-flops is T -T_{setup} - T_{prop} -T_{skew}. Hence, the minimal latency λ_{uv} for flip-flop based pipelined wire (*u,v*) is give by:

$$\lambda_{uv} = \left\lceil \frac{t_{uv}}{T - T_{setup} - T_{prop} - T_{skew}} \right\rceil - 1 \qquad (1)$$

When latches are employed instead, signals can depart from a latch at any time in the active interval rather than a single moment. The latches can be properly inserted in a way such that signals always arrive at a latch at its active interval with sufficient safety margin for setup time and clock skew, i.e. in the interval of [T -T_p,T -T_{setup}-T_{skew}], as shown by the yellow shaded interval in Figure 3 for positive level sensitive latches. Here T_p is the duration of positive clock signal level in one cycle. Then the only delay overhead in addition to the interconnect delay will be the propagation delay T'_{prop} of a latch. Therefore, the minimal feasible latency λ'_{uv} for latched based pipelined wire (*u,v*) is:

$$\lambda'_{uv} = \left\lceil \frac{t_{uv}}{T - T'_{prop}} \right\rceil - 1 \tag{2}$$

Please note that T'_{prop} is usually smaller than the flip-flop propagation delay T_{prop}. Comparing Equation (2) with Equation (1), we can see that a latch based wire pipelining can achieve smaller latency than flip-flop based approaches.

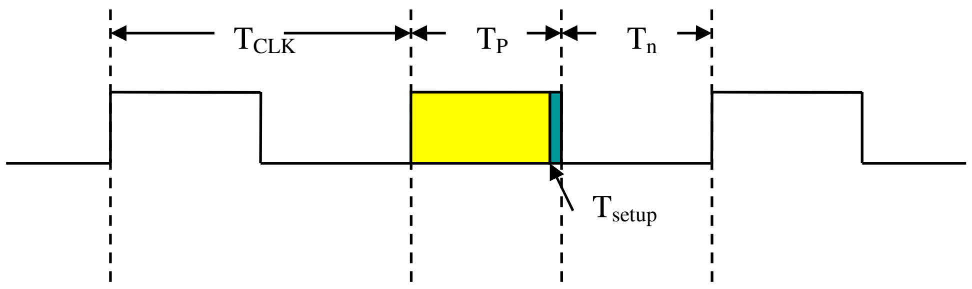

Fig. 3. Timing diagram for positive level sensitive latch.

Another great advantage of latch is its flexibility on timing constraint. As a latch allows signals to pass through in an interval, the path delay between two latches can be greater than one clock cycle provided that it is compensated by smaller path delays in its previous or next stage. This phenomenon is known as *cycle stealing* or *time borrowing*. In contrast, the path delay between two flip-flops cannot be greater than one clock cycle. The timing flexibility of latches is particularly appealing when obstacles exist for repeaters and synchronous elements. The obstacles can be hard macro, IP core or memory blocks which occupy certain region and disallow repeater or synchronous element insertions. In this scenario, the timing flexibility of latches can facilitate further improvement on latency compared with flip-flop based wire pipelining. Consider a two-pin net between two flip-flops *F1* and *F2* in Figure 4(a). Throughout this paper, we assume flip-flops are falling edge triggered and latches are positive level sensitive. There is an obstacle between spot *a* and *b*. The delays of the wire segments are *t1*, *t2* and *t3*, respectively, as indicated in Figure 4 and they satisfy $t1+t2$ Є $(T, T+T_p)$, $t2+t3$ Є $(T, T+T_p)$ and $t1+t2+t3$ Є $(T, 2T)$. Such an additive approximation can be used assuming that the interconnect is optimally buffered [7]. If only flip-flops are allowed in the pipelining, at least two flip-flops are needed as shown in Figure 4(b) to satisfy the constraint that a path delay is no greater than a clock cycle. If latch can be employed, one latch insertion, *L*, as in Figure 4(c) is sufficient for its timing constraint. Obviously, using latch results in less latency in this scenario.

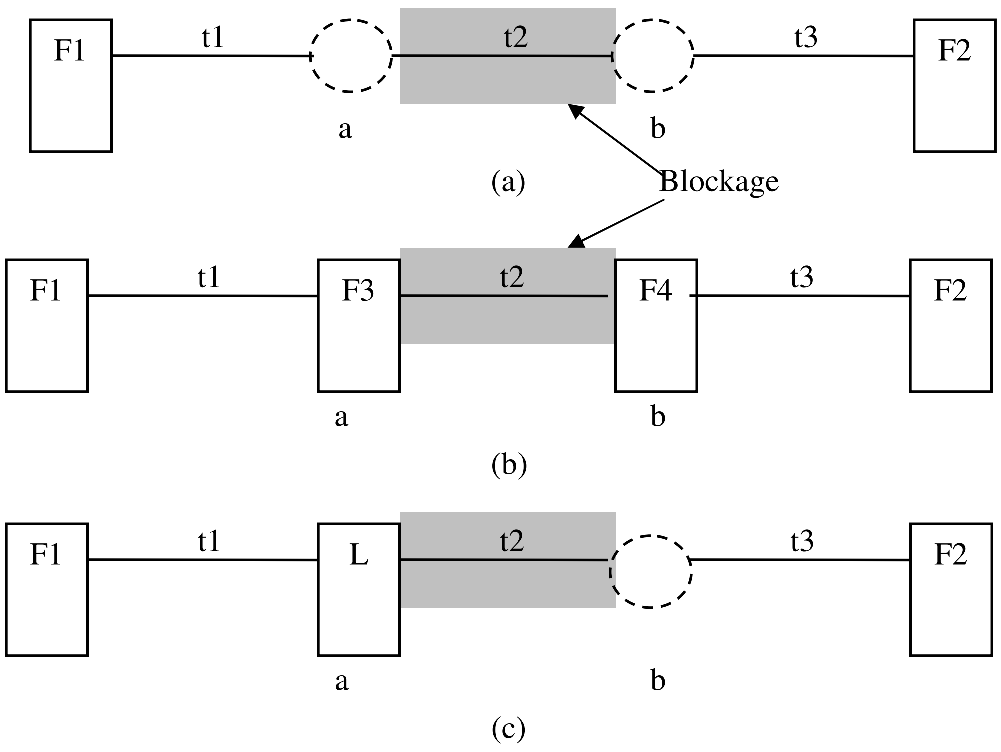

Fig. 4. Latency advantage of latch-based over flip-flop based wire pipelining. (a) Wire path with obstacles between a and b. (b) Flip-flop based wire pipelining. (c) Positive level sensitive latch based wire pipelining.

The timing flexibility of latches can also be utilized to reduce the number of synchronous elements inserted in wire pipelining. This is illustrated by the example in Figure 5, which is a multi-fanout net. In order to maintain functional correctness, the latency along each path is often required to meet certain constraint. In this example, the constraint is simply that the latency from the source to each sink has to be the same.

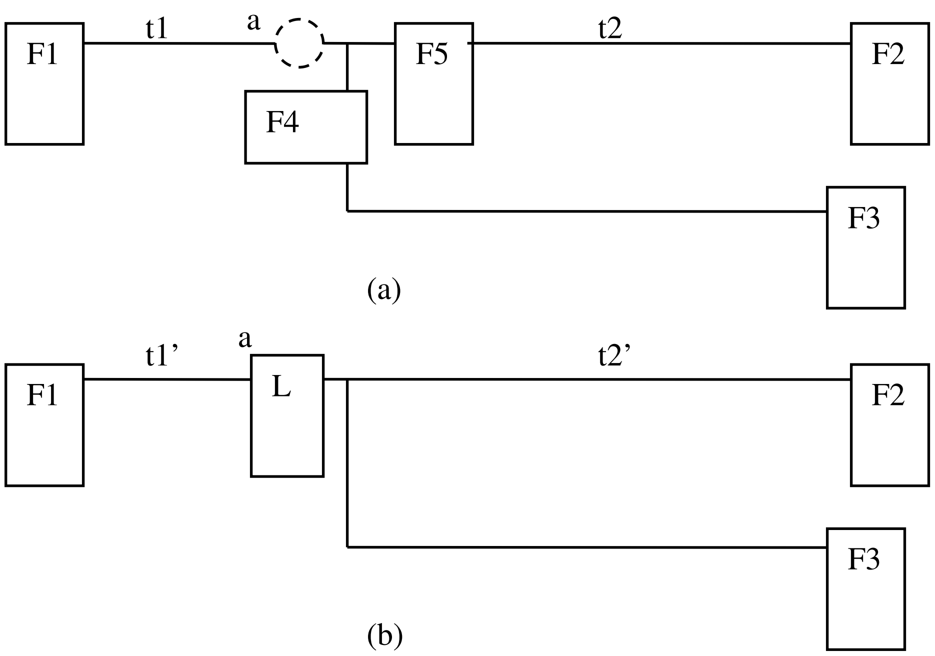

Fig. 5. Area advantage of latch-based over flip-flop based wire pipelining. (a) Flip-flop based wire pipelining. (b) Positive level sensitive latch based wire pipelining.

If *t2*, which is the delay from the branch point to flip-flop *F2*, is in the range of (T, $T + T_p$), two flip-flops need to be inserted to satisfy the equal latency constraint as shown in Figure 5(a). However, only one latch is necessary to meet the same constraint through cycle stealing as in Figure 5(b). Since a flip-flop is often as twice large as a latch, using latch in this scenario results in about 75% area savings.

2.2 CHALLENGES

Unfortunately, the timing flexibility of latches also brings extra design complexities. For a flip-flop, its output signal departure time is aligned to a clock signal switching edge and is independent of its input signal arrival time in general. For a latch, since there is a significant range of time for a signal to pass through, its output signal departure time directly depends on its input signal arrival time. This input-output timing coupling makes its timing constraints more complicated than those of flip-flops [6]. Moreover, the significantly long active interval increases the chance of short path constraint violation compared with flip-flops. Figure 6(a) shows a latch based pipelining for a multi-fanout net. The time diagram for the scenario is depicted in 6(b). Since the delay from *L2* to *L3* is small, the same signal indicated by the dashed lines is caught at both *L2* and *L3* in the same active interval as shown in Figure 6(b). This is a phenomenon of double clocking or short path violation [8]. In contrast, the short path violation is less likely to occur in a flip-flop based design, since the capture time interval for an edge-triggered flip-flop is very small, as depicted in Figure 6(c) and Figure 6(d). Therefore, sometimes delay padding [8] is necessary for latch based designs to correct short path violations. How to satisfy short path constraints with the minimum delay padding is a problem that needs crafted solutions.

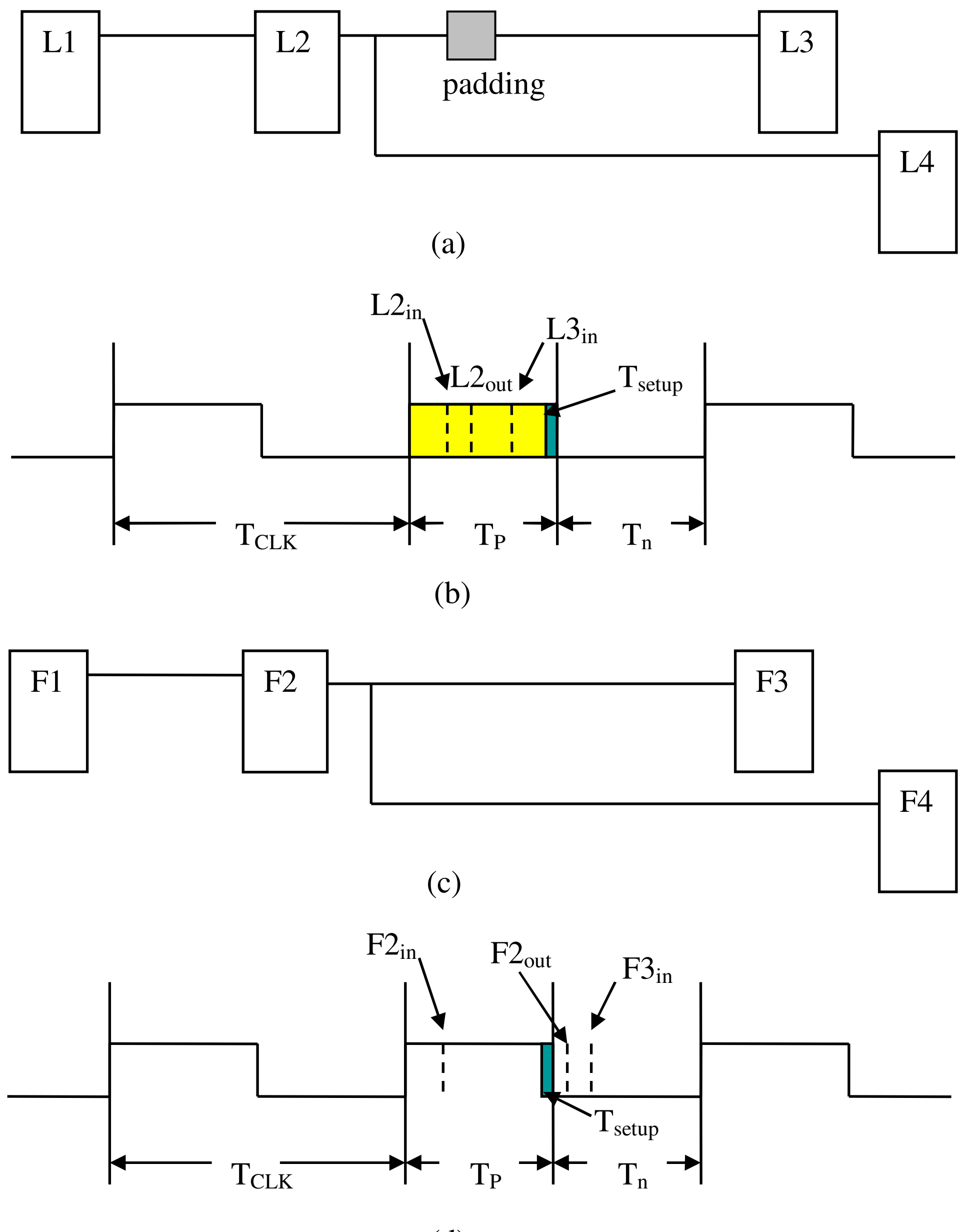

Fig. 6. Short path problem of latch based design. (a) Wire pipelining based on positive level sensitive latches. (b) Timing diagram for the pipelining without delay padding. (c) Wire pipelining based on falling edge-triggered flip-flops. (d) Timing diagram for the flip-flop based pipelining.

3 CONCURRENT REPEATER AND FLIP-FLOP INSERTION

The techniques of exploiting latches are integrated with the dynamic programming framework of [2], so this section gives an introduction of concurrent flip-flop and repeater insertion algorithms in [2]. Given a Steiner tree T with candidate insertion points and a repeater library G, including at least one flip-flop, the work of [2] proposes two variants of wire pipelining algorithm: (1) *MiLa* which searches for the minimum latency achievable for the Steiner tree; (2) *GiLa* which aims to minimize area cost while given latency constraints are satisfied. Both algorithms perform concurrent flip-flop and repeater insertion on the Steiner tree in a dynamic programming based framework like in [1, 9]. A flip-flop is also called clocked repeater in this work.

Wire is modeled as distributed RC, with resistance across the length of the wire and capacitance divided between the two ends of the wire. Wire delay is estimated with the Elmore delay model. The delay of a clocked or a non-clocked repeater, g_k, is expressed as $delay(g_k, c_{out})$ where c_{out} is the capacitive load.

The algorithms proceed from the sinks toward the source and candidate solutions, which are called *covers*, are generated and propagated in the bottom-up procedure. Each cover is associated with a node n_i in the tree and is expressed as a 4-tuple:

$$\gamma_i = (c_i, r_i, \lambda_i, a_i) \tag{3}$$

In the above 4-tuple, c_i is the downstream capacitance seen at n_i, r_i is the required arrival time, λ_i is the latency and a_i is the downstream repeater assignment. The covers at sinks are given and they are propagated to their parent nodes by the operation of *wire*. At a node for candidate repeater insertion, new covers are generated by the operation of *repeat*. Two sets of covers at a double branch node are merged with the operation of *merge*, in which two covers are joined through the operation of *join*. In the process of cover propagation, inferior covers are pruned out, using *Property 1*, to save runtime.

The pruning of the inferior covers, without compromising optimality, is based on an extension of the pruning introduced in [1]. Figure 7 shows *Property 1* that is used to determine the inferiority of one cover against the other.

Property 1 (Cover Inferirority): $\forall \gamma \in \Gamma, \gamma$ is inferior in Γ if $\exists \gamma' \in \Gamma$, such that at least one of the following is true:

a. $\lambda = \lambda', c \geq c', r < r'$;
b. $\lambda = \lambda', c > c', r = r'$;
c. $\lambda = \lambda', c = c', r = r'$, $\text{cost}(\gamma) > \text{cost}(\gamma')$;
d. $\lambda > \lambda', c \geq c', r \leq r'$;

Fig. 7. Cover inferiority.

In properties 1a and 1b (referred to as *nonmonotonic* inferiority rule) in Figure 7, it is obvious that γ is the inferior cover because with same latency it doesn't give a better required arrival time, and since its capacitance is also higher, its chances of giving a better solution later on during the bottom-up process is less. The property 1c (referred as *cover tie* inferiority rule) gives another dimension to the pruning, indicating that based on the user defined cost function, a solution is inferior if it gives higher cost, while all other attributes remain same. The property 1d (referred to as *extra latency* rule) follows from the same reasoning as for the properties 1a and 1b.

The pseudo-codes for the *merge* operation, the *MiLa* algorithm and the *GiLa* algorithm are given below in Figure [8-10], respectively. In the *Mila* algorithm, all the possible latency combinations are considered, at a double branch node, while merging the solutions of the left and right children, to get the set of solutions at the parent. This takes care that the covers that are propagated to the root are optimal. It implies that one of the children has to shift the latency of its solutions in all possible ways so that the solutions can be merged with all the solutions of its sibling.

In the *GiLa* algorithm, however, only those solutions, belonging to the left and right children, which have same latency, are merged. This implies that sometimes a procedure *ReFlop* may be performed to insert additional flip-flops to a branch, when covers from two branches are merged and their latency needs to be matched according to the latency constraints. The pseudo-code for the *ReFlop* operation is shown in Figure 11.

```
// Join covers with same latency from $\Gamma_u$ and $\Gamma_v$ in $\Gamma$.
merge($\Gamma_u$, $\Gamma_v$)
1.      // $\gamma_j^i \equiv$ i-th element of $\Gamma_j$ , $\lambda_j^i$ = latency of $\gamma_j^i$
2.      $\Gamma = \Phi$, x = y = 1
3.      while $x \leq |\Gamma_u|$ and $y \leq |\Gamma_v|$
3.1         if $\lambda_u^x > \lambda_v^y$ then y = y + 1, goto 3.
3.2         if $\lambda_u^x < \lambda_v^y$ then x = x + 1, goto 3.
3.3         $\Gamma = \Gamma \cup$ join($\gamma_u^x$, $\gamma_v^y$)
3.4         if $r_u^x \leq r_v^y$ then x = x + 1
3.5         if $r_v^x \leq r_u^y$ then y = y + 1
4.      return $\Gamma$
```

Fig. 8. Merge operation [2].

```
// Compute optimal covers $\Gamma_u$ of sub-tree $T_u$ rooted at
// node $u$, find minimum latency at each
// source-sink pair given repeater library $G$
$MiLa(T_u)$
1.      if $T_u$ is a leaf, $\gamma_u = (c_u, r_u, 0, 0)$
2.      else if node $u$ has one child node $v$ via edge $e_{u,v}$
2.1         $\Gamma_v = MiLa(T_v)$
2.2         $\Gamma_u = U_{\gamma \in \Gamma v} (wire(e_{u,v},\gamma))$    // Insert $|\Gamma_v|$ covers
2.3         $\Gamma_g = \Phi$
2.4         for each $g$ in $G$                 // Insert |G| covers
2.4.1           $\Gamma = U_{\gamma \in \Gamma u} (repeat(\gamma_{u,v},g))$
2.4.2           prune $\Gamma$          // Property 1
2.4.3           $\Gamma_g = \Gamma_g$ U $\Gamma$
2.5         $\Gamma_u = \Gamma_u$ U $\Gamma_g$
3.      else if $u$ has two child edges $e_{u,v}$ and $e_{u,z}$
3.1         $\Gamma_{u,v} = MiLa(T_{u,v})$, $\Gamma_{u,z} = MiLa(T_{u,z})$
3.2         // $\Gamma_{u,v} \equiv \{\Gamma^x, ..., \Gamma^y\}$, $\Gamma_{u,z} \equiv \{\Gamma^m, ..., \Gamma^n\}$
3.3         if $y < n$ then $swap(\Gamma_{u,v}, \Gamma_{u,z})$
3.4         for k = x – n to y – m //latency shift operation
3.4.1           $\Gamma_u = \Gamma_u$ U $merge(\Gamma^{m+k},\ldots, \Gamma^{n+k})$
4.      prune $\Gamma_u$    // Property 1
5.      if $u$ is source then Traverse the tree from root up and compute latency at
                    each sink
6.      return $\Gamma_u$
```

Fig. 9. MiLa algorithm [2].

```
// Compute optimal covers $\Gamma_u$ of sub-tree $T_u$ rooted at
// node $u$, given latency constraints $\lambda_u$ at each
// source-sink pair and repeater library $G$
$GiLa(T_u)$
1.        if $T_u$ is a leaf, $\gamma_u = (c_u, r_u, \lambda_u, 0)$
2.        else if node $u$ has one child node $v$ via edge $e_{u,v}$
2.1           $\Gamma_v = GiLa(T_v)$
2.2           $\Gamma_u = \cup_{\gamma \in \Gamma v} (wire(e_{u,v}, \gamma))$     // Insert $|\Gamma_v|$ covers
2.3           $\Gamma_g = \Phi$
2.4           for each $g$ in $G$                    // Insert |G| covers
2.4.1             $\Gamma = \cup_{\gamma \in \Gamma u} (repeat(\gamma_{u,v}, g))$
2.4.2             prune $\Gamma$          // Property 1
2.4.3             $\Gamma_g = \Gamma_g \cup \Gamma$
2.5           $\Gamma_u = \Gamma_u \cup \Gamma_g$            // $\Gamma_u \equiv \{\Gamma_x, ..., \Gamma_y\}$, $x,y$ indicate latency
2.6           if $u$ is source
2.6.1             if $x > 0$, exit: the net is not feasible
2.6.2             if $y < 0$, // insert $-y$ more flops in $\Gamma_u$
2.6.2.1               $\Gamma_u = ReFlop(T_u, -y)$
3.        else if $u$ has two child edges $e_{u,v}$ and $e_{u,z}$
3.1           $\Gamma_{u,v} = GiLa(T_u, v)$, $\Gamma_{u,z} = GiLa(T_u, z)$
3.2           // $\Gamma_{u,v} \equiv \{\Gamma_x, ..., \Gamma_y\}$, $\Gamma_{u,z} \equiv \{\Gamma_m, ..., \Gamma_n\}$
3.3           if $y < m$ // insert $m-y$ more flops in $\Gamma_{u,v}$
3.3.1             $\Gamma_{u,v} = ReFlop(T_{u,v}, m-y)$
3.4           if $n < x$ // insert $x-n$ more flops in $\Gamma_{u,z}$
3.4.1             $\Gamma_{u,z} = ReFlop(T_{u,z}, x-n)$
3.5           $\Gamma_u = \Gamma_u \cup merge(\Gamma_{u,v}, \Gamma_{u,z})$
4.        prune $\Gamma_u$     // Property 1
5.        return $\Gamma_u$
```

Fig. 10. GiLa algorithm [2].

```
// Insert $n$ extra flops in branch rooted by sub-tree $T_u$
ReFlop($T_u$, n)
1.    Traverse the tree from $T_u$ up removing sets $\Gamma$ along the way and computing
      the number crossed_flops of the flops crossed until either leaf or branch of
      degree 2 is reached.
2.    Traverse the tree down back to $T_u$ generating new sets $\Gamma$ using the wire and
      repeat functions but this time forcing the insertion in the branch an exact
      number of flops equal to crossed_flops + $n$. In particular, flip-flops are
      equally spaced along the branch so as to equally distribute the extra
      positive slack introduced. If there are more flip-flops to be inserted than
      available locations, extra flip-flops are inserted in already occupied
      locations.
3.    return $\Gamma_u$
```

Fig. 11. ReFlop operation [2].

The *wire, repeat* and *join* operations are described in the following section, in context to the use of level sensitive latches along with flip-flops and buffers in the *MiLa* and *GiLa* algorithms.

4 WIRE PIPELINING USING LATCHES

4.1 OVERVIEW

The problem formulations of this work and top level algorithm framework are the same as those of [2]. One major difference is that latches are included in the repeater library. Because of this change, the basic algorithms of *wire*, *repeat* and *join* are modified to satisfy the long path and the short path constraints for latches. In particular, short path violations need to be fixed by delay padding. This work proposes delay padding techniques that correct short path violations in a constructive manner rather than in post processing. Flip-flops are not excluded from the repeater library, even though the usage of latches is advocated. As shown in Figure 6 of Section 2, flip-flops have higher immunity to short path violations. In Figure 6(a), if the delay between L2 and L4 is less than $T - T_{setup} - T_{skew} - T_{prop}$ so that cycle stealing is unnecessary for the path from L2 to L4, the short path violation between L2 and L3 can be avoided by replacing L2 with a flip-flop. By keeping both flip-flops and latches in the repeater library, we let the dynamic programming decide the best way to fix short path violation at a node: delay padding or using a flip-flop. The assumptions of this work include:

- The *wire* pipelining is in a context of ordinary flip-flop based *circuit* designs. Therefore, either flip-flops or primary I/O will be met if we trace the fanin or the fanout of a net.
- Flip-flops are falling edge triggered and latches are positive level sensitive.
- All flip-flops and latches for a net are controlled by the same clock signal.
- The reference point for time is aligned with each falling edge of the clock signal.
- Even though the clock skew can be handled by our algorithm, it is neglected in this work for simplicity of description.

4.2 HANDLING OF LONG PATH CONSTRAINTS

For flip-flops, the long path constraint requires that the maximum path delay between two flip-flops should be no greater than T - T_{setup}. If there is a path between a driving flip-flop and a receiving flip-flop, the required arrival time at the input of the receiving flip-flop is normally set to T -T_{setup} and the required arrival time at the output of the driving flip-flop is required to be non-negative.

For latches, the maximum path delay allowable depends on if there is cycle stealing at its next stage. If there is no cycle stealing at its next stage, the maximum path delay can be greater than T - T_{setup} as long as it is no greater than T +T_p. However, a path delay greater than T -T_{setup} implies cycle stealing at current stage. If the amount of cycle stealing is t_{steal} , i.e., path delay of current stage is $t = T - T_{setup} + t_{steal}$, then the maximum path delay of its previous stage is bounded by $T - T_{setup} - t_{steal}$. In Figure 12, the long path constraint is illustrated in term of required arrival time. If we consider to insert a latch at node u, the required arrival time at the output of the latch is r_u which can be negative. A negative value of r_u implies cycle stealing at the stage and $-r_u = t_{steal} + T_{setup}$. Because of the cycle stealing, the required arrival time at the input of the latch is $r_{u,in} = T + r_u$.

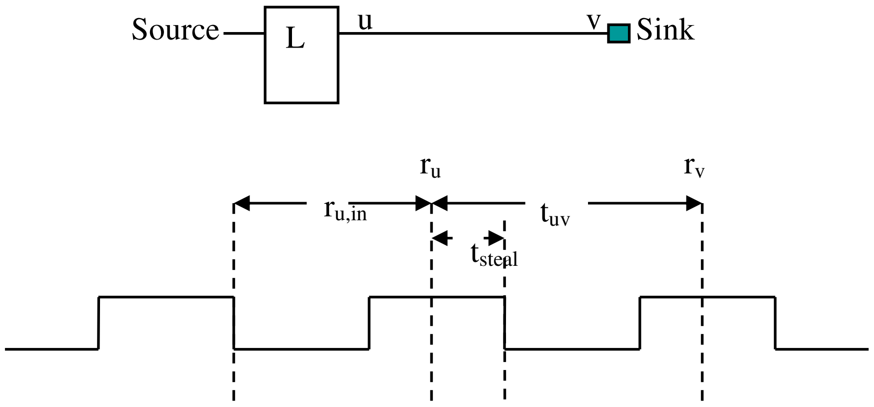

Fig. 12. Example of long path constraint for postive level sensitive latch based pipelining.

4.3 HANDLING OF SHORT PATH CONSTRAINTS

Unlike in flip-flop based designs in which the chance of short path violation is often negligible, the significantly long active interval for latches allows greater chance of short path violations. The short path constraint for latch based designs is presented in [11]. Consider the case that signals propagate from latch L_i to another latch L_j, as shown in Figure 13. The signal arrival time at latch L_j is required to be $a_j \geq T_{hold,\ j}$. By expressing a_j in term of departure time d_i, propagation delay $T_{prop,i}$ and $t_{i,\ j}$, we can obtain the short path constraint for $t_{i,\ j}$ as follows.

$$t_{i,j} \geq T_{hold,j} - T_{prop,i} - d_i \qquad (4)$$

$$d_i = \max(a_i, T - T_P) \qquad (5)$$

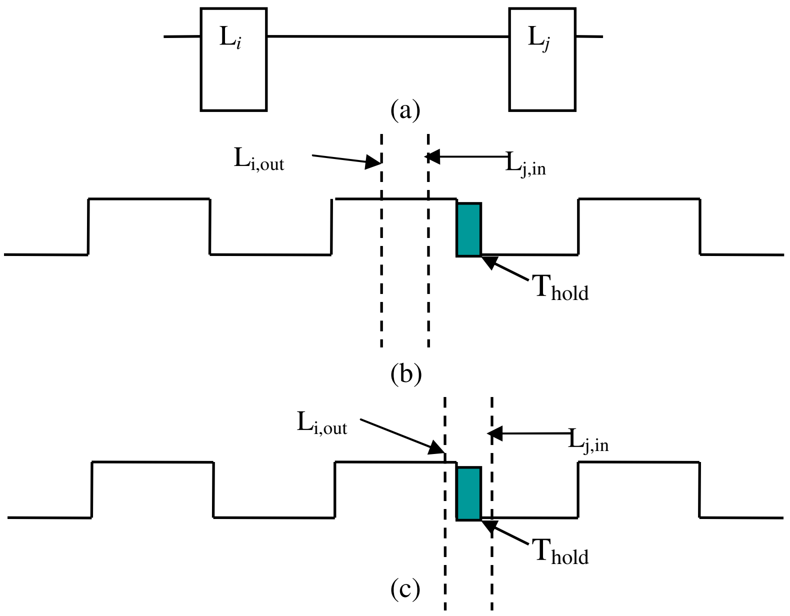

Fig. 13. Dependency of short path violation on the previous stage delay. (a) Wire pipelining based on positive level sensitive latches. (b) Signal arrival at L_i is close to $-T_P$. (c) Signal arrival at L_i is close to 0.

The troublesome part of this constraint is its dependence on the signal departure time d_i which in turn depends on the signal arrival time at latch L_i. In other words, a same path delay may or may not cause short path violation depending on its signal arrival time. This can be illustrated through the example in Figure 13. In Figure 13, the path delay between latch L_i and L_j is $t_{i,j} < T_p$. Whether or not this delay may cause short path violation depends on the signal arrival time at latch L_i. If that delay between L_i and its previous stage is small and d_i is close to $-T_p$ as shown in Figure 13(b), a short path violation occurs. However, if the path delay between L_i and its previous stage is large such that the signal departure time d_i at L_i is close to 0 as indicated in Figure 13(c), the small $t_{i,j}$ causes no short path violation. This dependence is particularly troublesome for the bottom-up dynamic programming based approach [2], as the signal departure time of previous stage is not known in the *repeat* operation at a node.

4.3.1 Post Processing

In traditional latch based circuit designs, the short path problem is usually solved by delay padding in a post processing procedure [8]. For each path with delay less than T_p, signal arrival time of its previous stage is known in a post processing and short path violations can be identified easily. However, the early work of [8] considered only gate delay and neglected wire delay which is dominating gate delay in modern technology.

When wire delay is also included, it is hard for the post processing technique like [8] to handle the case of multi-fanout trees. If a delay element is padded in any child branch, through either wire snaking or capacitance padding, the delay in the sibling may be increased due to the additional capacitive load, leading to a long path violation in that sibling. For example, in Figure 6(a), delay $t_{2,3}$ between *L2* and *L3* is less than T_p and causes short path violation, while delay $t_{2,4}$ between *L2* and *L4* is nearly $T + T_p$. If a delay element is padded between *L2* and *L3* through either wire snaking or capacitance padding, the delay to *L4* may be increased due to the additional capacitive load. Consequently, a long path violation may be induced between *L2* and *L4*. Therefore, the short path constraint needs to be considered together with the long path constraint in a constructive manner.

4.3.2 Uniform Delay Padding

One observation is that if the path delay between two latches is greater than T_p, then the short path constraint is guaranteed to be satisfied. Therefore, a simple method for short path violation corrections is to increase a path delay to T_p whenever it is less than T_p. Note that the path has to start with a latch, but can end with either a latch or a flip-flop. This method is called *uniform delay padding*. Even though short path violations can be eliminated completely through this method, it is conservative in a sense that some delay padding may be unnecessary. As explained before, short path violation will not happen if the signal arrival time of previous stage is large, even when the path delay is less than T_p. Hence, the uniform delay padding may cause extra cost unnecessarily. Moreover, unnecessary delay padding may increase delay of critical path as described in previous section and thereby degrade the latency along the critical path.

4.3.3 Deferred Delay Padding

To avoid the pessimism in the uniform delay padding, this work proposes a new delay padding heuristic that defers the actual padding until it is clearly necessary. Considering signals propagating from latch L_i to latch L_j, if the propagation delay is less than T_p by τ, there is potential short path violation depending on the signal arrival time at L_i. Instead of padding delay of τ immediately, we just record it as *Potential Delay Padding* (PDP) without doing actual padding. Only when the algorithm proceeds to a moment that the arrival time at L_i is known, part of or the entire amount of the PDP τ is indeed padded. The real delay padding procedure is called *Instantiate-Padding* in which the padding cost is increased. Traditionally, the required arrival time (RAT) means the latest or the upper bound of the arrival time. To facilitate the deferred delay padding, the earliest required arrival time also needs to be considered. Thus, we specify RAT with both an upper bound r and a lower bound $\underline{r}$, i.e., $\underline{r} \leq RAT \leq r$. The traditional RAT actually refers to r. Hence, two more factors $\underline{r}$ and $\underline{\tau}$ are kept for a cover, i.e.

$$\gamma_i \equiv (c_i, r_i, \lambda_i, a_i, \underline{r}_i, \tau_i) \qquad (6)$$

For a sink node j, its RAT must satisfy $\underline{r}_j = r_j - T \leq RAT \leq r_j$ and its PDP $\tau_j = 0$. If the parent node of j is node i and the delay between them is $t_{i,j}$, then the RAT at i is updated to $\underline{r}_i = \underline{r}_j - t_{i,j} \leq RAT \leq r_i = r_j - t_{i,j}$.

The pseudocodes for the *cover* operations of *wire, repeat* and *join* are given below in Figure [14-16]. These operations handle the long path constraints for latches and flip-flops, and short path constraints for latches. The terms $R_{u,v}$ and $C_{u,v}$ represent the resistance and capacitance, respectively, of edge $b_{u,v}$.

```
// Wire Operation
wire($b_{u,v}$,$\gamma_u$)
1.      $\gamma = \Phi$
2.      if $slack = r_v - R_{u,v}(C_{u,v}+c_v) \geq -T_p$
2.1         $\underline{r}_{u,v} = \underline{r}_v - R_{u,v}(C_{u,v} + c_v)$
2.2         $\gamma_{u,v} = (2C_{u,v} + c_v, slack, \lambda_v, 0, \underline{r}_{u,v}, \tau_v)$
3.      return $\gamma_{u,v}$   //end wire function
```

Fig. 14. Wire operation.

The *wire* operation updates the two limits of the required arrival time according to the delay introduced by the wire insertion. The latency and PDP fields remain unchanged.

```
// Repeat Operation
repeat($\gamma_{u,v}$,$g$)
1.          $\gamma = \Phi$
2.          if $slack = r_{u,v}\text{-}delay(g,c_{u,v}) \geq -T_p$
2.1             $\underline{r}_u = \underline{r}_{u,v} - delay(g,c_{u,v})$
2.2             if $g$ is not clocked
2.2.1               $\gamma = (load(g), slack, \lambda_{u,v}, g, \underline{r}_u, \tau_{u,v})$
2.3             else
2.3.1               $(\underline{r}'_u, slack', \tau_u)$ = Deferred-Delay-Padding($\underline{r}_u$, $slack$, $\tau_{u,v}$)
2.3.2               if $g$ is flip-flop
2.3.2.1                 if $slack' \geq 0$
2.3.2.1.1                   $\gamma = (load(g), T - T_{setup}, \lambda_{u,v}+1, g, 0, 0)$
2.3.3               else if $g$ is latch
2.3.3.1                 if $slack' \geq -T_p$
2.3.3.1.1                   $r_u = min(T - T_{setup}, T + slack')$
2.3.3.1.2                   $\gamma = (load(g), r_u, \lambda_{u,v}+1, g, \underline{r}'_u, \tau_u)$
3.          return $\gamma$    //end repeat function
```

Fig. 15. Repeat operation.

The *repeat* operation updates the initial solution according to the type of repeater inserted, i.e. buffer, flip-flop or latch. For buffer the latency and PDP remains unchanged, while for flip-flop the latency increases by 1 and the PDP still remains unchanged. This is because flip-flops don't require short path fix. However, for latch, the latency increases by 1 and the short path constraints are verified by the *Deferred-Delay-Padding* function. This function decides if actual padding is to be done or not, and the amount by which the earliest RAT and PDP need to be modified. In either case of clocked repeater insertion, flip-flop or latch, the long path constraints are checked before the repeater insertion.

```
// *Join* Operation
$join(\gamma_{u,v},\gamma_{u,z})$
1.      $r_u = min(r_{u,v}, r_{u,z})$
2.      $\underline{r}_u = max(\underline{r}_{u,v}, \underline{r}_{u,z})$
3.      $\tau_u = max(\tau_{u,v}, \tau_{u,z})$
4.      if ($\underline{r}_u > r_u$)
4.1         $r'_u$ = Instantiate-Padding($\underline{r}_u - r_u$)
4.2         update load capacitance $c'_{u,v}$ and $c'_{u,z}$, $\underline{r}'_u = r'_u$
5.      $a_u = a_{u,v}\ U\ a_{u,z}$ // Unite repeater assignment
6.      $\gamma_u = ((c'_{u,v}+c'_{u,z}), r'_u, max(\lambda_{u,v},\lambda_{u,z}), a_u, \underline{r}'_u, \tau_u)$
7.      return $\gamma_u$    //end *join* function
```

Fig. 16. Join operation.

In the *join* operation, the latest required arrival time r_i of a node i is obtained by taking the maximum among its child branches and the earliest required arrival time r_i is obtained from the minimum among its child branches. If $\underline{r}_i > r_i$, it implies that it is impossible to implement the *join* without violating the short path constraints. To correct the short path violation, $\underline{r}_i$-r_i amount of delay is padded at the short branch.

The pseudocode of *Deferred-Delay-Padding* function, used in *repeat* operation is given below in Figure 17. The procedure updates PDP (τ), $\underline{r}$ and determines if the PDP should be instantiated and how much of PDP need be instantiated. If a PDP is instantiated, the latest required arrival time r will also be updated. The main idea is to remove the short path violations by performing minimum delay padding to reduce the cost.

```
// Deferred-Delay-Padding operation
// Input: Candidate solution at a node
// Output: Updated solution ($\underline{r}'$, $r'$, $\tau'$)
Deferred-Delay-Padding($\underline{r}$, r, $\tau$)
1.      if $\underline{r} \le .-T_p$
1.2         $\underline{r}' = 0$; $\tau' = 0$ // clear PDP
2.      else if $\underline{r} \le 0$
2.1         $\underline{r}' = \underline{r}+T$
2.2         if $\tau == 0$
2.2.1           $\tau' = \underline{r}+T_p$ // a new PDP
2.3         else
2.3.1           $\tau' = \min(\underline{r}+T_p, \tau)$ // update PDP
3.      else
3.1         if $\underline{r} > \tau$
3.1 1           $r'$ = Instantiate-Padding($\tau$)
            /* $d$ is delay between current node and the downstream latch,
            and $\underline{r}_v'$ is $\underline{r}$ at the downstream latch, after delay padding*/
3.1.2           Deferred-Delay-Padding($\underline{r}_v'$ - d, $r'$,0)
3.2         else
3.2.1           $\underline{r}' = T$
3.2.2           $\tau' = \tau-\underline{r}$ // update PDP
3.2.3           $r'$ = Instantiate-Padding ($\underline{r}$)
4.      return ($\underline{r}'$, $r'$, $\tau'$)
```

Fig. 17. Deferred delay padding operation.

The procedure of *Deferred-Delay-Padding* and other cover operations is illustrated through two examples. The first example deals with a single path while the second example is about multi-fanout branches. In both examples, the clock period is $T = 8$, the active interval for latch is $T_p = 4$ and the RAT of sinks are assumed to be $0 \le RAT \le 8$. Also the propagation delay through a latch is neglected for the simplicity of the description.

In Figure 18, if a latch is inserted at node b, $r_b = -2$ which is in the range of $(-T_p, 0)$ and therefore a new PDP $\tau' = 2$ is generated. Since the delay between b and c is $t_{b,c}$ which is less than T_p by 2, this PDP 2 is an upper bound of delay padding for the path from *b* to *c*. When the algorithm proceeds to a moment that node *a* is considered for latch insertion, we get $\underline{r}$ = -4 ≤ -T_p. Therefore, the PDP of 2 is cleared to zero based on the *Deferred-Delay-Padding* procedure. The timing diagram of Figure 18(b) also explains why the PDP of 2 is cleared. The earliest departure time from latch at *a* is -T_p = -4, even though the arrival time at *a* can be significantly earlier than -4. Even for this earliest departure time, the signal arrival time to latch at *b* is 6 which will not cause short path violation without delay padding.

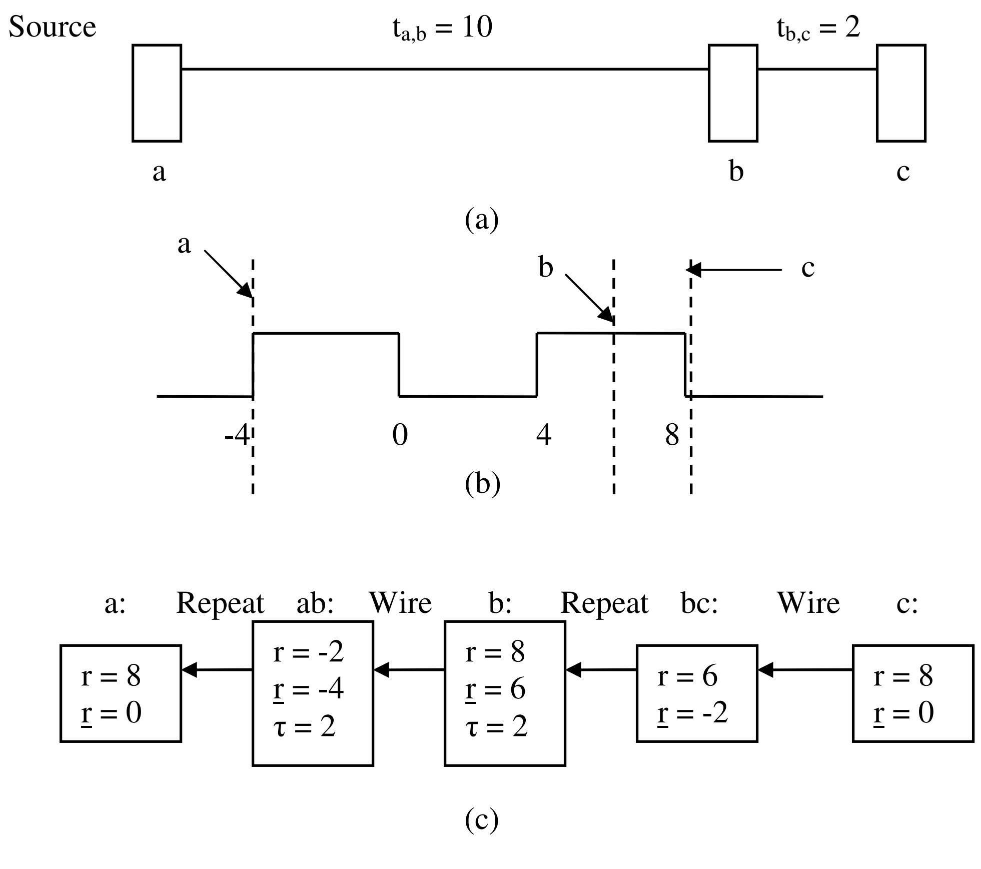

Fig. 18. Example of deferred delay padding along a path. (a) Positive level sensitive latch based wire pipelining example. (b) Timing diagram for the path. (c) Cover computation for the path.

In Figure 19, when branches $e_{c,d}$ and $e_{c,e}$ are joined at node c, $r_c = \min(r_{c,d}, r_{c,e}) = -2$ and $\underline{r}_c = \max(\underline{r}_{c,d}, \underline{r}_{c,e}) = -1$. Since $r_c < \underline{r}_c$, a short path correction is necessary and $\underline{r}_c - r_c = 1$ unit of delay is padded in the branch $e_{c,d}$ by the *Instantiate-Padding* operation. Next, $t_{b,c}$ is increased to $t'_{b,c} = 1$ due to the increase of capacitance on $e_{c,d}$. After the *wire* operation for edge $e_{b,c}$, the earliest required arrival time $\underline{r}$ is -3. If a latch is inserted at node b through the *repeat* operation, PDP $\tau = 1$ is induced. When we proceed to considering latch insertion at node a, the earliest required arrival time $\underline{r}$ is 2 which is greater than PDP $\underset{\cdot}{\tau} = 1$. Therefore, another unit of delay is padded on edge $e_{c,d}$ based on the *Deferred-Delay-Padding* procedure. Even though the delay padding at $e_{c,d}$ may increase the delay from b to e, this increase is less than the amount of PDP being instantiated in the short path. Thus, the long path constraint will not be violated by the delay padding.

In addition to the techniques developed for exploiting latches, this work suggests another change to the algorithms in [2]. In [2], when the latency of a cover γ is larger than the latency of another cover γ' and γ is inferior in terms of load capacitance and required arrival time, the cover γ will be pruned. This rule is called *extra latency inferiority rule*. This rule is modified such that γ will be pruned out only when $\gamma > \gamma', c \geq c', r \leq r'$ and γ is not the only cover with latency of λ. In other words, if there is only one cover for a specific latency, this cover will not be pruned out. When covers from two branches are merged, less latency discrepancy will happen between the two branches with the application of the *modified extra latency inferiority rule*. Therefore, the number of calls on the *ReFlop* procedure is also reduced.

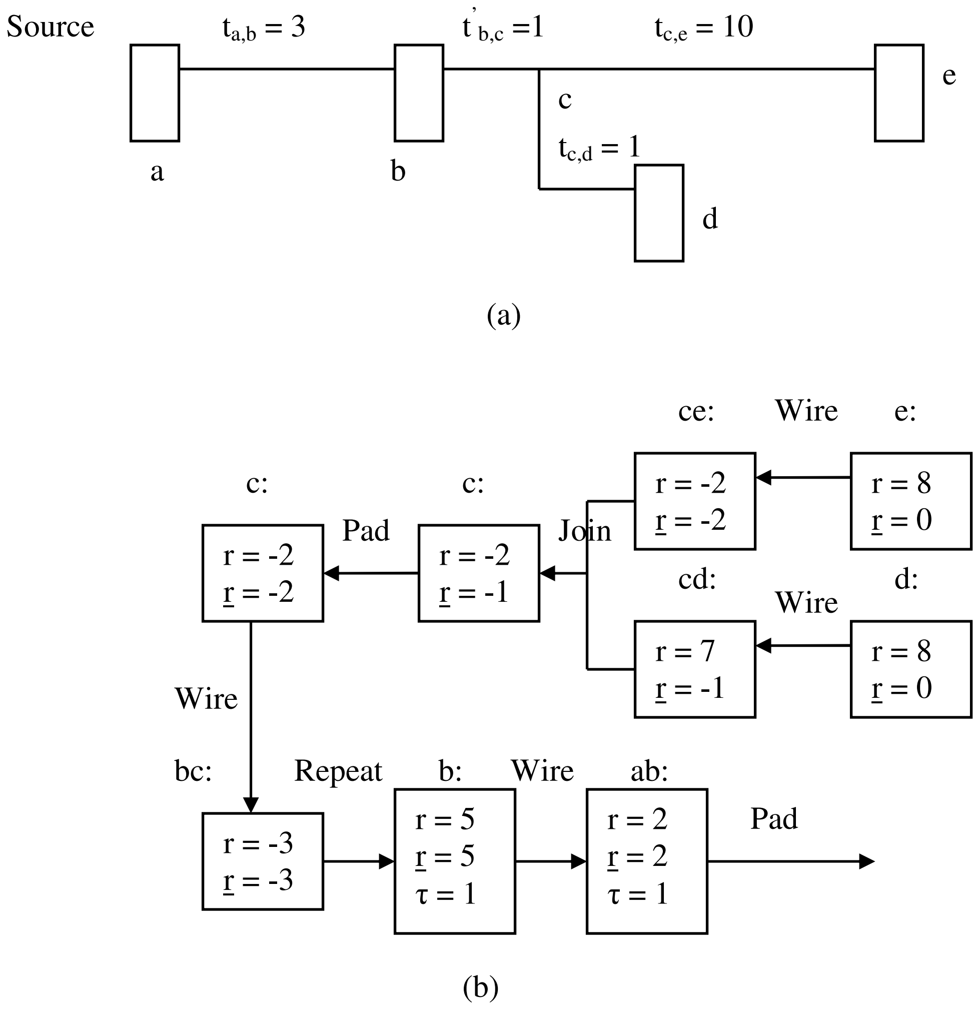
Source
$t_{a,b} = 3$
$t'_{b,c} = 1$
$t_{c,e} = 10$
a
b
c
$t_{c,d} = 1$
d
e
(a)
ce:
Wire
e:
c:
c:
Pad
Join
$r = -2$, $\underline{r} = -2$
$r = 8$, $\underline{r} = 0$
$r = -2$, $\underline{r} = -2$
$r = -2$, $\underline{r} = -1$
cd:
d:
Wire
$r = 7$, $\underline{r} = -1$
$r = 8$, $\underline{r} = 0$
Wire
bc:
Repeat
b:
Wire
ab:
Pad
$r = -3$, $\underline{r} = -3$
$r = 5$, $\underline{r} = 5$, $\tau = 1$
$r = 2$, $\underline{r} = 2$, $\tau = 1$
(b)

Fig. 19. Example of deferred delay padding for nets with branches. . (a) Positive level sensitive latch based wire pipelining example with branches. (b) Cover computation for the net.

4.4 ALGORITHM COMPLEXITY

To understand the time complexity of the above algorithms, consider the case where there is one buffer in the repeater library, but there are no clocked repeaters (flip-flop or latch). In such a scenario, the *MiLa* reduces to the nonclocked repeater problem in [1]. As reported in [1], the time complexity of the buffer (nonclocked repeater) insertion problem in [1] is $O(|B|^2)$, where $|B|$ is the number of candidate locations, or single/double branch nodes, in the routing tree. Thus, to find the time complexity of *MiLa*, we need to find the effect of adding clocked repeaters (flip-flop and latch) to the repeater library *G*, on the runtime of problem in [1].

Now consider the case of interconnects, where repeater library *G* also contains a single flip-flop. The analysis of the size of the cover sets in [15] shows that under pruning operation using *Property 1* and the assumption, that the sum of the capacitance of a wire between two contiguous candidate repeater locations and the input capacitance of any non-clocked repeater in G is greater than the input capacitance of any clocked repeaters in G, the cover set at a node can have covers with at most three different latencies, i.e. for cover set $\Gamma_u = \{ \Gamma_u^{k}, \Gamma_u^{k+1}, \ldots, \Gamma_u^{k+n-1} \}$, the maximum value of *n* is 3. In case of adding a latch to the repeater library G, the size of cover set is not affected because in a given scenario of clocked repeater insertion, either a latch or a flop is inserted at a candidate location, and not both of them.

Thus, for the case when repeater library G has non-clocked repeaters, a flip-flop and a latch, the time complexity does not increase. Particularly, this is due to the fact the increase in size of every cover set Γ_u after repeater insertion is still $O(|G|)$, where $|G|$ is the number of repeaters in the repeater library *G*. This gives the time complexity for *MiLa* to be $O(|G|^2.|B|^2)$.

5 EXPERIMENTAL RESULTS

The experiments are carried out on a SUN Sparc Ultra-80 workstation with four 450MHz CPUs and 4Gb RAM. Eleven nets with 1-17 sinks are generated for the testing. The clock period is $5ns$.The wire resistance and capacitance are 0.126Ω and $0.139fF$ per unit length, respectively. Only one non-inverting repeater, one flip-flop and one latch are included in the repeater library. They all have output resistance of 300Ω and input capacitance of $5fF$. The intrinsic delay is $10ps$ for the repeater and latch, and is $20ps$ for the flip-flop. The setup time for both the latch and the flip-flop is $10ps$.

The experiments are designed to test: (1) if there is latency advantage of using latches in *MiLa*; (2) if there is area advantage of using latches in *GiLa*; (3) if the deferred delay padding can satisfy the short path constraint with less area cost than the uniform delay padding. These are tested in two scenarios of with and without obstacles. The scenario without obstacles is the same as in [2] such that the candidate insertion sites are evenly distributed. If repeater obstacles are considered, there may be larger gap between two neighboring candidate insertion sites, or the number of candidate locations may be different. The description of the test cases is given in Table I below.

The *MiLa* results without obstacles are shown in Table II. Among the 11 nets, the latency is reduced by using latches for four times. Due to the discrete nature of latency, latency reduction of one implies saving of one clock cycle. Reducing the latency for the nets being processed also decreases the number of synchronous elements needed for other nets to maintain functional correctness.

In Table III for the *MiLa* results with obstacles, there are two cases where no feasible solutions can be found for flip-flop based method. However, feasible solutions can be obtained by using latches. The CPU time in seconds is also listed in Table II and Table III. We can see the runtime is very fast for all these cases.

TABLE I
TEST CASES USED FOR THE EXPERIMENTS

Net #	Without Obstacles		With Obstacles	
	# Sinks	# Candidates	# Sinks	# Candidates
1	1	2	1	3
2	2	4	2	5
3	3	5	3	5
4	4	7	4	7
5	5	14	5	13
6	5	14	5	13
7	8	17	8	18
8	9	30	9	29
9	10	30	10	28
10	15	36	15	35
11	17	40	17	40

TABLE II
MiLa RESULTS WITHOUT OBSTACLES

Net #	Only FF		FF + Latch	
	Latency	CPU Time	Latency	CPU Time
1	1	0.00	1	0.01
2	2	0.01	2	0.02
3	2	0.00	1	0.02
4	2	0.01	1	0.03
5	2	0.00	2	0.02
6	2	0.02	2	0.01
7	2	0.03	2	0.04
8	4	0.04	3	0.03
9	3	0.02	3	0.03
10	2	0.05	2	0.07
11	3	0.05	2	0.07

TABLE III
MiLa RESULTS WITH OBSTACLES

Net #	Only FF		FF + Latch	
	Latency	CPU Time	Latency	CPU Time
1	2	0.00	1	0.01
2	1	0.01	1	0.01
3	2	0.01	1	0.02
4	1	0.00	1	0.02
5	2	0.01	2	0.01
6	3	0.03	2	0.02
7	2	0.02	2	0.04
8	4	0.04	3	0.05
9	X	0.02	3	0.05
10	2	0.04	2	0.05
11	X	0.04	3	0.06

The *GiLa* results are shown in Table IV and V. The nets and candidate insertion sites are the same as in Table I. The latency constraints for these nets are from the previous *MiLa* results. In several cases, only using flip-flops may not find feasible solutions satisfying the latency constraints. However, the constraints can be met by using latches. For those feasible solutions, only the number of elements inserted and the area cost are reported here. The area value in parenthesis indicates the ratio with respect to the area from using latches with the deferred delay padding. Using latches can always yield less area cost except for the case that net 8 is optimized with the uniform delay padding. This exception is due to the tighter latency constraint for using latches according to the *MiLa* result. However, the cost for net 8 is less than the flip-flop based solution when we apply the deferred delay padding with latches. The results also indicate that using latches is even more helpful when there are obstacles. For each feasible flip-flop based solutions, its area cost is 11%-125% greater than that of using latches with the deferred delay padding.

TABLE IV
GiLa RESULTS WITHOUT OBSTACLES

Net	Only FF			FF+Latch Uniform Delay				FF+Latch Deferred Delay			
#	#Buf	#FF	Area	#Buf	#FF	#Latch	Area	#Buf	#FF	#Latch	Area
1	0	1	0.6	0	0	1	0.3	0	0	1	0.3
2	0	3	1.8	1	0	3	1.2	1	0	3	1.2
3	1	4	2.7	2	0	2	1.2	2	0	2	1.2
4	2	5	3.6	3	0	3	1.8	3	0	3	1.8
5	8	6	6.0	9	1	3	4.2	9	0	4	3.9
6	10	3	4.8	10	1	2	4.2	10	0	3	3.9
7	X	X	X	12	0	4	8.4	12	0	4	4.8
8	14	14	12.6	16	1	11	15.9	18	0	10	8.4
9	X	X	X	18	1	10	9.0	18	0	11	8.7
10	30	5	12.0	31	0	4	10.5	31	0	4	10.5
11	29	8	13.5	35	0	4	11.7	35	0	4	11.7

TABLE V
GiLa RESULTS WITH OBSTACLES

Net	Only FF			FF+Latch Uniform Delay				FF+Latch Deferred Delay			
#	#Buf	#FF	Area	#Buf	#FF	#Latch	Area	#Buf	#FF	#Latch	Area
1	0	2	1.2	1	0	1	0.6	1	0	1	0.6
2	2	2	1.8	3	0	1	1.2	3	0	1	1.2
3	1	4	2.7	2	0	2	1.2	2	0	2	1.2
4	X	X	X	3	0	3	1.8	3	0	3	1.8
5	7	6	5.7	8	1	3	3.9	8	0	4	3.6
6	8	4	4.8	9	2	1	4.2	9	1	2	3.9
7	X	X	X	14	0	4	5.4	13	0	4	5.1
8	13	14	12.3	15	1	11	15.6	17	0	10	8.1
9	X	X	X	15	0	12	8.1	15	0	12	8.1
10	29	5	11.7	31	0	4	10.5	31	0	4	10.5
11	X	X	X	34	1	4	12.0	34	0	5	11.7

When a net is small, the advantage of using deferred delay padding vs. the uniform delay padding is not strong as there is less chance of the short path problem. For larger nets, the results in Table IV and Table V show that the deferred delay padding can further reduce area cost. The CPU time for latch based *GiLa* is about the same as those in Table II and Table III. Therefore, the extra computation cost of the proposed techniques is trivial.

6 CONCLUSIONS

This work demonstrates the advantages of using level sensitive latches in wire pipelining, along with flip-flops, with a timing complexity of the algorithm being quadratic in number of candidate repeater locations and number of repeaters in the repeater library. These advantages are due to the less setup time overhead and particularly the timing flexibility that allows cycle stealing. By using latches, both the latency and the area cost can be improved. New techniques, especially deferred delay padding, are proposed to overcome the difficulties in latch based approaches. Experimental results support the advantages of using latches and the effectiveness of the proposed algorithms in reducing the latency and area of latent interconnects.

REFERENCES

[1] L. P. P. P. V. Ginneken, “Buffer placement in distributed RC-tree networks for minimal elmore delay,” *IEEE International Symposium on Circuits and Systems*, 1990, pp. 865–868.

[2] P. Cocchini, “Concurrent flip-flop and repeater insertion for high performance integrated circuits,” *IEEE/ACM International Conference on Computer Aided Design*, 2002, pp. 268–273.

[3] S. Hassoun, C. J. Alpert, and M. Thiagarajan, “Optimal buffered routing path constructions for single and multiple clock domain systems,” *IEEE/ACM International Conference on Computer Aided Design*, 2002, pp. 247–252.

[4] C. Lin and H. Zhou, “Retiming for wire pipelining in system-on-chip,” *IEEE/ACM International Conference on Computer Aided Design*, 2003, pp. 215–220.

[5] L. Zhang, Y. Hu, and C. C. P. Chen, “Wave-pipelined on-chip global interconnect,” *TAU Workshop*, 2003, pp. 46–51.

[6] 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, no. 3, pp. 322–333, 1992.

[7] R. H. J. M. Otten and R. Brayton, “Planning for performance,” *ACM/IEEE Design Automation Conference*, 1998, pp. 122–127.

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

[9] J. Lillis, C. K. Cheng, and T. Y. Lin, “Optimal wire sizing and buffer insertion for low power and a generalized delay model,” *J. Solid State Circuits*, vol. 31, pp. 437–447, Mar. 1996.

[10] V. De and S. Borkar, "Low power and high performance design challenges in future technologies," in *Great Lakes Symposium VLSI*, Mar. 2000, pp. 1–6.

[11] K. A. Sakallah, T. N. Mudge, and O. A. Olukotun, "checkTc and minTc: Timing verification and optimal clocking of synchronous digital circuits," *IEEE/ACM International Conference on Computer Aided Design*, 1990, pp. 552–555.

[12] Dennis K. Y. Tong, and Evangeline F. Y. Young, "Performance-driven register insertion in placement," *ACM/IEEE International Symposium Physical Design*, 2004, pp. 53-60.

[13] Vidyasagar Nookala, and Sachin S. Sapatnekar, "A method for correcting the functionality of a wire-pipelined circuit," *ACM/IEEE Design Automation Conference*, 2004, pp. 570–575.

[14] P. Saxena, N. Menezes, P.Cocchini and D.A. Kirkpatrick, "Repeater scaling and its impact on CAD," *IEEE Transactions on Computer Aided Design of Integrated Circuits and Systems*, vol. 23, no. 4, pp. 451–463, 2004.

[15] P.Cocchini, "A methodology for optimal repeater insertion in pipelined interconnects," *IEEE Transactions on Computer Aided Design of Integrated Circuits and Systems*, vol. 22, no. 12, pp. 1613–1624, 2003.

VITA

Vikram Seth was born in Lucknow, India in 1978. He completed his Bachelor of Technology degree in electrical engineering from the Indian Institute of Technology, Kanpur, India in May 2000. He subsequently worked for three years as an ASIC Design Engineer before starting his graduate studies as a computer engineering major at Texas A&M University in the fall of 2003. His research interests are in ASIC Design and Computer Aided Design for VLSI physical design. He can be reached at the following email address: vikram.seth@gmail.com. His permanent address is c/o Mr. Sumitro Samaddar, 10341 Tonita Way, Cupertino, CA 95014.