Spec-based flip-flop and latch repeater planning

Man Chung Hon
Intel Corportation
Santa Clara, CA 95054, USA
manch.c.hon@intel.com

Abstract— Shrinking process geometries and frequency scaling give rise to an increasing number of interconnects that require multiple clock cycles. This paper explores efficient techniques to insert flip-flops and latches to meet pre-determined latency and margin constraints at the receivers. Previous approaches push timing margins to either ends of interconnect. We present an $O(n \log n)$-time algorithm to insert flip-flops that evens out timing margins across the entire interconnect, resulting in more robust designs and faster design convergence. An $O(n \log n)$-time extension to handle symmetric, two-phases latches is also presented. Experimental results verify the correctness and practicality of our approach.

I. Introduction

Deep submicron process has created a number of new design challenges. Primary among them is the increasing dominance of interconnects. Current scaling trends show that the frequency of high-performance ICs approximately doubles and the die size grows by 25% with every process generation. Interconnect optimization techniques such as repeater insertion and gate sizing have proven effective in reducing interconnect delay. However, with such short clock cycles, the delay on global signals may be longer than one clock cycle even *after* being optimized with these techniques. Insertion of clocked repeaters such as flip-flops and latches become necessary on these signals.

In a typical design flow, microarchitects determine the target latencies for each driver-receiver pair in the design during the early phase of design cycle. Traditionally, flip-flops and latches are coded into RTL manually. It is difficult for the circuit designers to try out different configurations and placements. A recent study on the effect of process scaling on repeaters [9] shows that, not only is a clock cycle's worth of metal length shrinking faster than the scaling of geometries, it is also decreasing at a much faster rate than the repeater-to-repeater distance. It is predicted that at the 45-nm process node, as many as one in every four repeaters is clocked [9].

There has been growing interests among researchers on the problem of multi-cycle global interconnects. Lu, Zhong, Koh and Chao [8] proposed an analytical formulation for the simultaneous insertion of flip-flops and buffers to minimize latencies. Hassoun, Alpert and Thiagarajan [3] combined routing, buffer and flip-flop insertion in multiple clock domain systems. Both [8] and [3] only work with two-pin nets, and do not respect latency constraints specified *a priori*. Cocchini [2] extended van Ginneken's classical dynamic programming structure [12] to simultaneously insert flip-flops and buffers, and can be directed to either minimize latencies or to meet specified latency constraints. This algorithm was in turn extended by Seth, Zhao and Hu [10] to incorporate single phase level-sensitive latches. As pointed out by Akkiraju and Mohan [1], Cocchini's approach puts all margins, both positive and negative, to the first segment right after the driver. (On a 2-pin net bounded by two flip-flops, margin is defined to be the difference between a clock cycle and the arrival time at the receiving flip-flop. On a multi-fanout net, we take the minimum from all the receivers.) Instead, [1] proposed another extension of van Ginneken to insert flip-flops in such a way that margins are evenly distributed in the driver and the receiver ends, while holding the middle segments timing-tight. Researchers have also investigated techniques other than clocked repeaters. Zhang, Hu and Chen [14] suggested a new global interconnect architecture using the wave-pipelining technique. Lin and Zhou [7] applied retiming to improve the clock speed on an initial flip-flop placement.

In this paper, we present an algorithm to insert flip-flops to even out margins throughout the entire interconnect. This way of margin distribution is often what a circuit designer wants intuitively. In the case of *positive* margin, putting all at the extremities of the interconnect, as is done in Cocchini [2] and Akkiraju and Mohan [1], makes the middle segments timing-critical. A large number of tight segments could pose challenges to downstream design stages. This is especially problematic when designers seek to improve the chip's speed in minor revisions of a taped-out design. The middle segments, which barely pass the original frequency goal, may now come in with negative margins. In the case of *negative* margin, putting all at the ends of the interconnect makes the timing problem more difficult to fix. As an example, imagine a 2-pin net with 9 flip-flops in between, and a negative margin of -100 pico-seconds. Fixing 10 nets each with a -10 ps margin is arguably easier than fixing one net with a -100 ps margin (Cochinni), or two nets with -50 ps margin

0-7803-9451-8/06/$20.00 ©2006 IEEE.

(Akkiraju and Mohan)

The remainder of the paper is organized as follows. Section II sets up the flip-flop and latch insertion problems in terms of equivalent retiming problems. Section III presents an efficient $O(n \log n)$-time flip-flop insertion algorithm that evens out margins. Section IV extends the flop algorithm to handle 2-phase symmetric latches, which also runs in $O(n \log n)$ time. Both algorithms are asymtotically faster and easier to implement than previous work. Section V presents experimental results, and we conclude in Section VI.

II. Preliminaries

In this paper, we model the topology of an interconnect as a tree $T(V, E)$. $V = \{v_d \cup S_N \cup S_{TN}\}$ is a set of n nodes, with root v_d and leaves S_N of T being the interconnect driver and receivers respectively. The intermediate Steiner nodes S_{TN} contain candidate locations for flop and latch insertion. E is a set of $|V| - 1$ edges corresponding to wires. A *stage* is an ordered subset of edges $(u, u_1), \ldots, (u_k, v)$ in E such that u is either v_d or contains a flop (latch), v is either in S_N or contains a flop (latch), and $u_1, \ldots, u_k$ are free of flops and latches. Each receiver $v_r \in S_N$ has a non-negative, integeral latency requirement $l(v_r)$, and a non-negative required margin $m(v_r)$. $l(v_r)$ specifies the number of flops, or in the case of latches, half the number of latches, on the unique path from v_d to v_r. The margin requirement states that the signal must arrive at v_r no later than $m(v_r)$ time-units before the falling clock edge. We model the delay inside a flop or a latch by a single, fixed real number $d_{cell} \geq 0$.

Our modeling of wire delay follows [1]. We assume the delay of a wire to be proportional to the square root of its RC-content. The maximum unrepeated distance for a wire was computed based on performance and reliability requirements. Buffers were inserted based on this distance and simulations were done to compute the delay across the wire. The delays were then curve-fitted to a linear equation with respect to the square root of the wire's total RC-content. This allows us to quickly estimate the wire delay $d(e)$, under the assumption that the wire is part of a buffered interconnect path.

We now describe our notion of proper latch timing. We follow closely the definition in [4]. In particular, we require the latches to hold the same values as in an identical circuit in which all elements, including both gates and wires, have *zero* propagation delay. The notion of proper latch timing is *structural* in nature, in the sense that the circuit should operate correctly regardless of the functions it computes.

Formally, we seek to solve the following two problems:

1. **Flip-Flop Repeater Insertion Problem** Assign flops to S_{TN} such that: (i) margins are equally distributed among all stages; and (ii) latency and margin requirements at the receivers are satisfied.

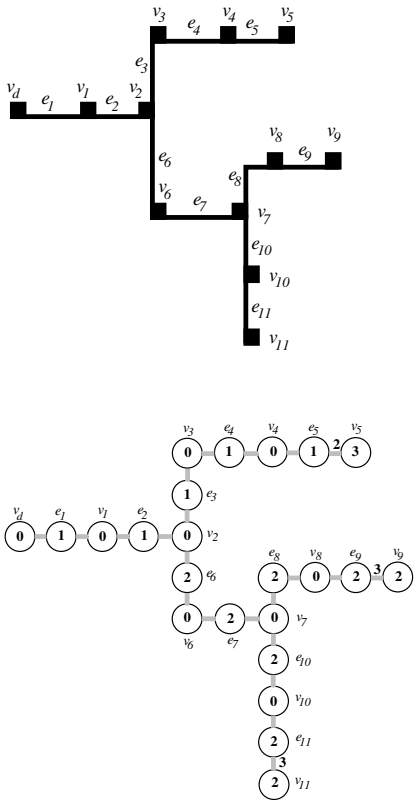

Fig. 1. Top: Original graph model $T(V, E)$. $d(e_1), \cdots, d(e_5) = 1$, $d(e_6), \cdots, d(e_{11}) = 2$. Receivers have required margins $m(v_5) = 3, m(v_9) = m(v_{11}) = 2$, and required latencies $l(v_5) = 2, l(v_9) = l(v_{11}) = 3$. Bottom: Transformed graph model $T'(V', E')$. Edges have implicit weight 0, unless otherwise specified. Numbers on nodes are associated delays.

2. **Latch Repeater Insertion Problem** Assign two-phase, symmetric level-sensitive latches to S_{TN} such that: (i) latency and margin requirements at the receivers are satisfied; and (ii) timing is structurally correct.

Let us break the circuit T into $|v_r|$ sub-circuits T_r, each consisting of the unique 2-pin sub-net from the driver v_d to a single receiver v_r. It is clear that distributing margins evenly on the 2-pin net T_r amounts to minimizing the clock period of T_r. The clock period of T is given by the relation $\text{period}(T) \geq \max_{v_r} \text{period}(T_r) \geq \text{period}(T_r)$ for all v_r. Minimizing clock period of T evens out margins by proxy.

Retiming [5] is a technique that relocates registers to reduce cycle time while preserving circuit functionality. As a first step, we convert each edge $e \in E$ into a vertex with propagation delay $d(e)$. Formally, we transform $T(V, E)$ into another tree $T'(V', E')$, with $V' = V \cup E$ and each edge $u \xrightarrow{e} v$ in the original edge set E spawns two edges in E': $u \rightarrow e$ and $e \rightarrow v$.

There are three possibilities for $d(v')$, $v' \in V'$. If v' comes from a receiver v_r in the original vertex set V,

$d(v') = m(v_r)$. If v' comes from an edge $e \in E$, $d(v') = d(e)$. For all other vertices, $d(v') = 0$.

As in [5], sequential elements are represented implicitly by a non-negative integral *weight* $w(e')$ on $e' \in E'$. In our case, $w(e') = 0$ except for the edges that are adjacent to a receiver v_r, in which case $w(e') = l(v_r)$ ($2l(v_r)$ for latches) (Fig. 1.) Note that the original flop or latch placement, if there is any, is ignored. Our algorithms operate on the transformed tree model T'. For ease of exposition, we will refer to both as T in the rest of the paper. Note that the asymptotic bounds derived for T' also hold for T.

A retiming solution can be viewed as an integral labeling r of vertices. $r(v)$ represents the number of sequentials moved from v's outputs toward its inputs. In particular, if we require that $r(v_d)$ at the driver and $r(v_r)$ at all receivers to be *equal*, it is guaranteed that latencies at the receivers remain the same after retiming. Since we model delays across a clocked repeater as a constant d_{cell}, it does not factor in the selection of one retiming solution over another. For the rest of the paper we assume $d_{cell} = 0$.

III. Flip-Flop Insertion

A general framework to calculate the minimum period is to perform a binary search over a range of possible clock periods until a user-definied accuracy is reached. At each step in the binary search, a new clock period is tried. Retiming is performed to recalculate $r(v)$ at each node in an attempt to make the circuit work with the current clock period. The smallest period for which retiming succeeds is returned as the best clock period.

The efficiency of the binary search depends strongly on the retiming step in its inner loop. One algorithm for the retiming step is the relaxation method, modelled after the Bellman-Ford algorithm for the single source shortest path problem [5, 11]. Shown in Fig. 2, relaxation works on general graphs. We denote by $\delta(v)$ the latest arrival time at v along any *combinational* path that terminates at v. It is obvious that the clock period c is given by $c = max_{v \in V} \delta(v)$. For a given set of $r(v)$, we calculate $\delta(v)$ for all v, which takes $O(E)$ time. The relaxation method thus runs in $O(VE)$ time.

An analysis by Yen [13, 6] explains why the $V-1$ loop in lines 2-7 of Graph_Flop_Clk_Feas (Fig. 2) works. The latest arrival time $\delta(v)$ converges to its final value if the edges along the longest path (in terms of propagation time) from the source node v_d to v are relaxed in order. The sequence of edges relaxed in the loop consists of $V-1$ copies of some ordering of E, and therefore contains every vertex-disjoint path as a subsequence. Since T is a tree, there is a unique path from v_d to v. There is thus no need for the $V-1$ loop. Instead, we combine the calculation of $\delta(v)$ with the relaxation of r in a single topological tour of the nodes. The new linear-time relaxation algorithm is shown in Fig. 2.

We bound the binary search in Fig. 2 from above with

```
Graph_Flop_Clk_Feas(G,r,c)
/* Input:  Graph G(V,E), label r, period c
 * Output:  Feasibility of c */
1   for each v ∈ V  r(v) ← 0
2   for |V| − 1 {
3       Compute retimed edge weights
4       Compute δ(v) for all v ∈ V
5       for all v ∈ V such that δ(v) > c
6           r(v) ← r(v) + 1
7   }
8   Compute retimed edge weights
9   if max_{v∈V} δ(v) > c
10     then no feasible retiming
11     else the current r is legal

Tree_Flop_Clk_Feas(T,r,c)
/* Input:  Tree T(V,E), label r, period c
 * Output:  Feasibility of c */
1   for each v ∈ V  r(v) ← 0
2   for v ∈ V in topological order {
3       Compute δ(v)
4       if δ(v) > c
5          then r(v) ← r(parent(v)) + 1
6               δ(v) ← d(v)
7          else r(v) ← r(parent(v))
8   }
9   Compute retimed edge weights
10  if max_{v∈V} δ(v) > c
11     then no feasible retiming
12     else the current r is legal

Tree_Flop_Insert(T,ε)
/* Tree Flip-Flop Insertion
 * Input:  Tree T(V,E), rel error ε
 * Output:  Retiming label r*/
1   d_max ← max_{v∈V} d(v)
2   c_hi ← max_{v_r} d(v_d ⇝ v_r)
3   c_lo ← max_{v_r} {d(v_d ⇝ v_r)/(l(v_r) + 1)}
4   r ← 0
5   while c_hi − c_lo > ε * d_max {
6      c ← (c_hi + c_lo)/2
7      if Tree_Flop_Clk_Feas(T,r,c) = true
8      then c_hi ← c
9      else c_lo ← c
10  }
11  return r
```

Fig. 2. Clock feasibility test in FF retiming for general graphs, clock feasibility test for trees and FF insertion for trees

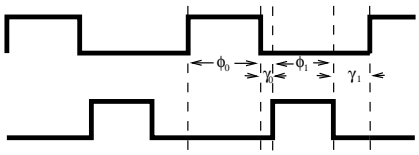

Fig. 3. Two phase clocking scheme, reproduced from [4].

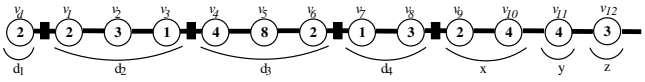

Fig. 4. The path $v_4 \rightsquigarrow v_8$ satisfying Condition 1 implies $v_5 \rightsquigarrow v_7$ meets the same condition. d_i denotes the sum of vertex delays in the i-th stage.

$c_{hi} = max_{v_r} d(v_d \rightsquigarrow v_r)$. This corresponds to the circuit being entirely combinational. The search is bounded from below by $c_{lo} = max_{v_r}\{d(v_d \rightsquigarrow v_r)/(l(v_r)+1)\}$, which corresponds to margins being distributed perfectly evenly along every driver to receiver path. Note that for $c >= c_{lo}$, it is guaranteed that Tree_Flop_Clk_Feas leaves $r(v_r) = 0 = r(v_d)$ for all receivers v_r: the latency requirement is satisfied throughout the binary search. Tree_Flop_Insert solves Problem 1 in $O(V \log V)$ time for any fixed ϵ.

IV. Latch Insertion

In a two-phase clocking scheme $\langle \phi_0, \gamma_0, \phi_1, \gamma_1 \rangle$, two clocking phases are employed (Fig. 3). ϕ_0, ϕ_1 denote the duty cycle of the first and second phases, and γ_0, γ_1 denote the gaps. The period is given by $\phi_0+\gamma_0+\phi_1+\gamma_1$. A two-phase symmetric clocking scheme is characterized by equal duty cycles and gaps: $\phi_0 = \phi_1 = \phi$ and $\gamma_0 = \gamma_1 = \gamma$.

The conditions for proper timing of T are based on considering the operation of T when all propagation delays are 0. It can be shown that T is properly timed if and only if for every path $u \overset{p}{\rightsquigarrow} v$, the following condition is satisfied [4]:

$$d(p) \leq c\left(\frac{1+w(p)}{2}\right) + \phi. \tag{1}$$

Furthermore, we only need to make sure (1) is satisfied for *maximal* paths. These are paths that start right after a latch and ends right before another. To see this, note that the right hand side of (1) stays the same as we take in more nodes into the path, as long as the number of latches does not change. On the other hand, more nodes could potentially increase $d(p)$, leading to a tighter inequality. To reduce clutter on the formulas, we set $\phi = 0$ for the rest of the paper.

(1) leads to a greedy algorithm. Nodes are visited in topological order staring with the driver. In Fig. 4, suppose all the maximal paths up until v_{10} satisfy (1), and we are considering whether a latch *must* be inserted between v_{10} and v_{11} to maintain timing feasibility. Applying (1) to the maximal paths that end with v_{11}, we consider whether the following inequalities are satisfied:

$$\begin{aligned} y &\leq \frac{c}{2} - x \\ y &\leq \frac{c}{2} * 2 - d_4 - x \\ &\vdots \\ y &\leq \frac{c}{2} * 5 - d_1 - d_2 - d_3 - d_4 - x \end{aligned} \tag{2}$$

Case 1 Not satisfied. In this case we must insert a latch. x becomes the new d_5, the right hand sides of (2) adjusted, and a new inequality is added to the set:

$$\begin{aligned} z &\leq \frac{c}{2} - y \\ &\vdots \\ z &\leq \frac{c}{2} * 6 - d_1 - d_2 - d_3 - d_4 - d_5 - y \end{aligned} \tag{3}$$

Case 2 Satisfied. We can avoid adding a latch:

$$\begin{aligned} z &\leq \frac{c}{2} - x - y \\ &\vdots \\ z &\leq \frac{c}{2} * 5 - d_1 - d_2 - d_3 - d_4 - x - y \end{aligned} \tag{4}$$

The inequality satisfaction tests can be done more efficiently by replacing (2) with a single inequality $y \leq d_{min}$, where d_{min} is the minimum of the right hand sides in (2):

$$d_{min} = \min(\frac{c}{2} - x, \ldots, \frac{c}{2} * 5 - d_1 - d_2 - d_3 - d_4 - x).$$

Comparing the updated inequalities in (3) and (4) with (2), it is evident that d_{min} can be computed efficiently. Starting with $d_{min}(v_d) = \frac{c}{2}$ at the driver, $d_{min}(v)$ is computed recursively from $d_{min}(u)$, where $u \rightarrow v$ is an edge in E, as follows:

$$d_{min}(v) = \begin{cases} \frac{c}{2} + \min(0, d_{min}(u) - d(u)) \\ \quad \text{if } u \rightarrow v \text{ is latched} \\ d_{min}(u) - d(u) \text{ otherwise} \end{cases} \tag{5}$$

With (5), feasibility of any period c can be determined in linear time (Fig. 5). Using this as a subroutine, a binary search over the range of possible periods is performed. Let $D = max_{v \in V} d(v)$. The search is bounded by $|V| * D + 2 * \phi$ from above, and $2 * \phi$ from below. This results in an $O(n \log n)$-time algorithm (Fig. 5).

V. Experimental Results

We implemented three different spec-based sequential insertion algorithms in C++, and compared them on a block from an Intel Xeon^{TM}microprocessor designed on 90-nm process technology.

```
Tree_Latch_Clk_Feas(T,r,c)
/* Input:  Tree T(V,E), label r, period c
 * Output:  Feasibility of c*/
1   for each v ∈ V  r(v) ← 0
2   for v ∈ V in topological order {
3       Compute d_min(v) according to Eq 5
4       if v is source or sink
5          then r(v) ← 0
6          elseif d(v) > d_min(v)
7              then r(v) ← r(parent(v)) + 1
8              else r(v) ← r(parent(v))
9   }
10  for e = u → v ∈ E {
11      if w(e) − r(u) + r(v) < 0
12         then no feasible retiming
13  }
14  The current r is legal

Tree_Latch_Insert(T,ε)
/* Input:  Tree T(V,E), rel error ε
 * Output:  Retiming label r*/
1   d_max ← max_{v∈V} d(v)
2   c_hi ← |V| * d_max + 2 * φ
3   c_lo ← 2 * φ
4   r ← 0
5   while c_hi − c_lo > ε * d_max {
6      c ← (c_hi + c_lo)/2
7      if Tree_Latch_Clk_Feas(T,r,c) = true
8      then c_hi ← c
9      else c_lo ← c
10  }
11  return r
```

Fig. 5. Latch clock feasibility test for trees and latch insertion for trees

TABLE I
RESULTS OF VARIOUS SEQUENTIAL INSERTION ALGORITHMS. THE FLOP NUMBER OF *LATCH* IS HALF THE NUMBER OF LATCHES.

Algorithm	Flops	RunTime (seconds)
SBFIA	14989	3104.38
RFLOP	14907	11.54
LATCH	15028.5	11.92

- SBFIA - Spec-based FF insertion from [1].
- RFLOP - FF insertion algorithm from Section III.
- LATCH - Latch insertion algorithm from Section IV.

The design has 1769 multi-cycle nets, with fanouts ranging from 1 to 34. The nets' topologies are discretized by breaking the wires with a candidate insertion node every 100 microns. The relative error ϵ is set to 0.001. All experiments were run on a workstation with four Intel Xeon^{TM}2.8GHz/512KB processors and 8GB of memory, although none of the three algorithms exploit the parallelism afforded by the machine.

As shown in Table I, the three algorithms used similar amount of sequential resources. For accounting purpose a latch is counted as half a flop. Note that even though minimizing flop usage is not an explicit goal of *RFLOP*, it used slightly fewer flops than *SFBIA*, while *LATCH* used 0.26% more. A more detailed study of the result shows that *RFLOP* was more flop-efficient than *SBFIA* in every fanout bucket.

The comparison on runtime is more pronounced. Both *RFLOP* and *LATCH* finished under 12 seconds, while the $O(n^2 \log n)$-time *SFBIA* took over 50 minutes.

We observe that *stage spread* – the difference between maximum and minimum flop-to-flop propagation delay – is smaller for *RFLOP* than *SBFIA* (Table II), validating our design goal of a more even margin distribution. Errors from discretization and imbalance among multiple receivers keep the spread positive. Perfectly balanced margins would have given a spread of *zero*.

Both *RFLOP* and *LATCH* improved negative slacks compared to *SFBIA* (Table III). In the case of *LATCH*, all the negative slacks were wiped out. The median gain on frequency over *SFBIA* ranges from 15.96% to 35.11% for *RFLOP*, and 22.42% to 44.27% for *LATCH*.

VI. Conclusions

We have presented fast and practical algorithms for flop and latch insertion to facilitate design of pipelined interconnects within the latency constraints at the receivers. We argued that evening out timing margins across the entire interconnect is more beneficial than pushing them to either ends, and made it an explicit goal of our algorithms to distribute margins evenly. Experimental results

demonstrate the efficiency and efficacy of the algorithms on industrial test cases. unsrt

TABLE II
STAGE SPREAD IS MEASURED IN PICOSECONDS. THE NUMBER OF NETS IS GIVEN IN PARENTHESES NEXT TO THE FANOUTS.

Algorithm	No. Flops	Stage Spread	
		Median	Average
1-fanout (941)			
SBFIA	5766	162.12	175.96
RFLOP	5766	58.23	80.31
2-fanout (456)			
SBFIA	4968	249.78	253.46
RFLOP	4956	146.73	152.98
3-fanout (187)			
SBFIA	1877	269.94	256.91
RFLOP	1876	183.30	181.92
4-fanout to 6-fanout (145)			
SBFIA	1703	281.67	281.46
RFLOP	1671	205.34	200.73
7-fanout+ (40)			
SBFIA	675	333.72	319.72
RFLOP	638	230.98	224.37

TABLE III
NEGATIVE SLACK IS MEASURED IN PICOSECONDS. THE NUMBER OF NETS IS GIVEN IN PARENTHESES NEXT TO THE FANOUTS.

	Negative Slack			Freq Gain	
	Total	Worst	Nets	Median	Average
1-fanout (941)					
SBFIA	-38242.2	-213.4	707	0	0
RFLOP	-253.2	-16.6	192	17.51%	19.36%
LATCH	0.0	0.0	0	24.35%	29.67%
2-fanout (456)					
SBFIA	-17859.8	-220.0	413	0	0
RFLOP	-69.1	-6.3	59	17.47%	17.13%
LATCH	0.0	0.0	0	23.47%	25.53%
3-fanout (187)					
SBFIA	-7704.4	-180.8	168	0	0
RFLOP	-278.6	-15.0	44	15.96%	17.45%
LATCH	0.0	0.0	0	22.42%	26.48%
4-fanout to 6-fanout (145)					
SBFIA	-7756.8	-189.9	142	0	0
RFLOP	-335.8	-15.8	60	16.24%	21.66%
LATCH	0.0	0.0	0	24.05%	29.27%
7-fanout+ (40)					
SBFIA	-3312.1	-169.0	40	0	0
RFLOP	-119.4	-13.2	22	35.11%	32.36%
LATCH	0.0	0.0	0	44.27%	39.58%

REFERENCES

[1] N. Akkiraju and M. Mohan. Spec based flip-flop and buffer insertion. In *Proceedings of the 21st International Conference on Computer Design*, pages 270–275. IEEE Computer Society, 2003.

[2] P. Cocchini. A methodology for optimal repeater insertion in pipelined interconnects. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 22(12):1613–1624, 2003.

[3] S. Hassoun, C. J. Alpert, and M. Thiagarajan. Optimal buffered routing path constructions for single and multiple clock domain systems. In *Proceedings of the 2002 IEEE/ACM international conference on Computer-aided design*, pages 247–253. ACM Press, 2002.

[4] A. T. Ishii, C. E. Leiserson, and M. C. Papaefthymiou. Optimizing two-phase, level-clocked circuitry. *J. ACM*, 44(1):148–199, 1997.

[5] C. Leiserson and J. Saxe. Retiming synchronous circuitry. *Algorithmica*, 6(1):5–35, 1991.

[6] C. E. Leiserson and J. B. Saxe. A mixed-integer linear programming problem which is efficiently solvable. *J. Algorithms*, 9(1):114–128, 1988.

[7] C. Lin and H. Zhou. Retiming for wire pipelining in system-on-chip. In *Proceedings of the 2003 international conference on on Computer-aided design*, pages 215–220. IEEE Computer Society, 2003.

[8] R. Lu, G. Zhong, C. Koh, and K. Chao. Flip-flop and repeater insertion for early interconnect planning. In *Proceedings of the conference on Design, automation and test in Europe*, page 690. IEEE Computer Society, 2002.

[9] P. Saxena, N. Menezes, P. Cocchini, and D. Kirkpatrick. Repeater scaling and its impact on cad. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 23(4):451–463, 2004.

[10] V. Seth, M. Zhao, and J. Hu. Exploiting level sensitive latches in wire pipelining. In *Proceedings of the 2004 international conference on on Computer-aided design*, pages 283–290. IEEE Computer Society, 2004.

[11] N. Shenoy. Retiming: theory and practice. *Integr. VLSI J.*, 22(1-2):1–21, 1997.

[12] L. van Ginneken. Buffer placement in distributed rc-tree networks for minimal elmore delay. In *Proceedings of the IEEE International Symposium on Circuits and Systems*, volume 2, pages 865–868. IEEE Computer Society, 1990.

[13] J. Yen. An algorithm for finding shortest routes from all source nodes to a given destination in general networks. *Quarterly of Applied Mathematics*, 27(4):526–530, 1970.

[14] L. Zhang, Y. Hu, and C. Chen. Wave-pipelined on-chip global interconnect. In *Proceedings of the Design Automation Conference Asia and South Pacific*, page to appear. IEEE Computer Society, 2005.