Leveraging Delay Slack in Flip-flop and Buffer Insertion for Power Reduction

Lucanus Simonson, King Ho Tam, †Nataraj Akkiraju, †Mosur Mohan, Lei He *

EE Department, University of California, Los Angeles, CA
† Intel Corporation, San Jose, CA

Abstract

We show that the delay slack can be distributed optimally between flip-flops to reduce power in a pipelined interconnect, and such power reduction can be achieved by simultaneous flip-flop and buffer insertion satisfying latency and delay constraints specified at sinks. We develop a dynamic programming algorithm with effective pruning rules and pseudo polynomial time complexity with respect to the decimation and the length of a net. Experiments using a cluster of interconnect in a leading industrial high-performance design show that there exists plenty of useful slack for power reduction. Without jeopardizing the delay specification, as much as 17% of power can be saved for this cluster of interconnects.

1 Introduction

Increase in clock frequency has rendered buffered interconnect alone inadequate for meeting delay constraints for global interconnect. Flip-flop (FF) insertion has therefore become necessary for long interconnects that can no longer deliver signal in a single clock cycle, and more nets will require FF and buffer insertion to meet delay constraints as the technology scales. For a routing tree, let the latency of a node be the number of FFs between the source and the node, and the delay slack (or in short, slack) of a node be the difference between the clock period and the delay from the upstream FF to the node. Extending the dynamic programming algorithm originally developed for buffer insertion in an RC tree [8] , simultaneous FF and buffer insertion was studied for minimizing latency in [5] and for maximizing the minimum slack at the source and sinks while satisfying the specified latency in [1].

*This paper is partially supported by NSF CAREER award CCR-0306682, SRC grant HJ-1008, a UC MICRO grant sponsored by Analog Devices, Fujitsu Laboratories of America, Intel and LSI Logic, and a Faculty Partner Award by IBM. We used computers donated by Intel and SUN Microsystems. Address comments to lhe@ee.ucla.edu.

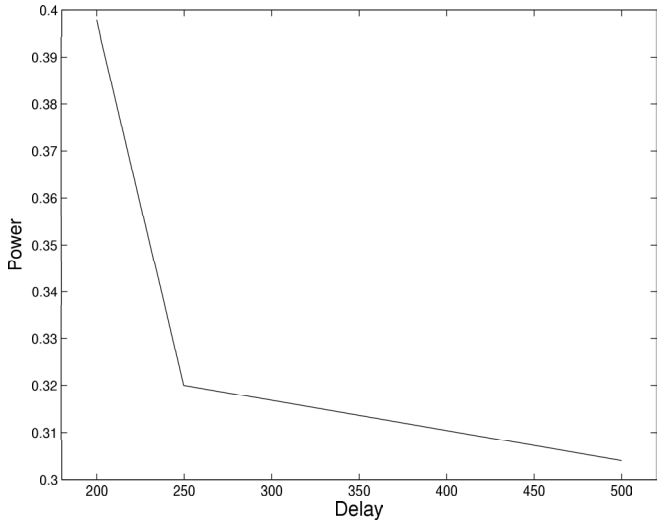

Figure 1. Power vs Delay Specification

Nowadays power constraints are also of increasingly critical importance to designers. Maximizing slack as in [1] may introduce unnecessary power dissipation. Figure 1 shows the power dissipation of a straight net of 2mm length with buffers inserted optimally to meet different delay specifications. One can clearly see that as we increase the delay budget for the line, the required power dissipation is reduced. If we distribute slack to intermediate FF stages as well, we may increase the delay budget for intermediate FF stages and in turn reduce the power needed by optimal buffer insertion as indicated by Figure 1. Such power optimal FF and buffer insertion has also been discussed at the full-chip level in [6]. However, the assumption of 2-pin net based on enumeration limits its use.

In this paper we formulate the simultaneous FF and buffer insertion problem for power optimization. Instead of minimizing latency and delay at the sinks as in [5], our formulation finds the FF and buffer insertion solution with minimum power but meeting the specified latency and delay at multiple sinks of a given routing tree.

We develop an efficient dynamic programming algorithm to properly distribute delay slack among interconnect segments for power reduction. We compare the solutions produced by our algorithm with those from [1], which will be referred to as **M**aximum **S**lack **D**ynamic **P**rogramming (MSDP). We call our formulation as **L**ow **P**ower **D**ynamic **P**rogramming(LPDP). We show that there exists plenty of useful delay slack in a high-performance industrial design, and our LPDP algorithm achieves an average of 17% power reduction compared to the MSDP algorithm.

The rest of this paper is organized as follows. We discuss modeling and problem formulation in Section II, and describe our algorithm in Section III. We present experimental results in Section IV and conclude in Section V.

2 Modelling and Problem Formulations

2.1 Models

We use the Π-type model in this paper. This model is known to accurately reflect the behavior of a distributed RC interconnect due to the fact that on average only half of the capacitance of a uniform wire lies downstream of a the resistance of a small segment. For buffers we use the simple first order model of an inverter, substituting the gate capacitance, drive resistance and drain capacitance for the nonlinear device, to reduce delay estimation for buffer insertion points to a form to which Elmore delay can be applied. Elmore delay has been shown to exhibit high fidelity [3] but is known to be an inaccurate first order estimation of RC delay and increasingly inadequate as it fails to capture the effect of inductance upon delay entirely. The Elmore model is particularly well suited to the bottom up dynamic programming approach we have chosen because it can be calculated by summing the product of resistance and downstream capacitance as the algorithm moves up the tree thereby enabling the computations of delays at each node. This is due to the iterative aspect of the calculation of Elmore delay, as can be seen in the equation 1 for the Elmore delay of a single line of n segments with iteration beginning at the end of the line and progressing to the beginning.

$$t_{Elmore} = \sum_{i=1}^{n}(R_i \sum_{j=1}^{i}(C_j)) \tag{1}$$

For the purposes of power estimation, spice simulation was used for leakage of buffers and FFs. Dynamic power per clock cycle for interconnect, buffers and FFs was modelled as the capacitive switching power for full Vdd to ground transition (2). In the absence of switching information we assume 0.15 switching probability, which is considered to be a reasonable value [4].

$$P_d = \frac{1}{2}(C_{clk} + 0.15C_{signal})V_{dd}^2 \tag{2}$$

2.2 Problem Formulation

There are three optimization objectives at each node. *Required arrival time* of a partial solution at node n, denoted $q(v)$, is defined as follows:

$$q(v) = \min_{u \in child(v)}(q(u) - delay(u)) - delay_{buf}(v) \tag{3}$$

where $q(u)$ is the required arrival time of a child partial solution u of v associated with a child node k of n. The delay due to the wire connecting u to v, denoted $delay(u)$, is defined as:

$$delay(u) = r(u)\left[m(u) + \frac{1}{2}c(u)\right] \tag{4}$$

where $r(u)$ is the resistance of the wire connecting the node k to the parent node n. Similarly $c(u)$ is the capacitance of the wire vu and $m(u)$ is the downstream capacitance of u. In (3) $delay_{buf}(v)$ is the gate delay of a buffer at node n driving capacitances at and downstream of n as follows:

$$delay_{buf}(v) = r_d(v)\left[c_d(v) + \sum_{u \in child(v)}(c(u) + m(u))\right] \tag{5}$$

In the case where v represents FF insertion at n, $q(v)$ is equal to the FF latency of solution v, denoted $f(v)$, multiplied by the clock period, t_{clk}, with the FF setup time, t_f subtracted out.

$$q(v) = f(v)t_{clk} - t_f \tag{6}$$

Downstream capacitance, denoted $m(v)$, is the second optimization objective, and is defined as:

$$m(v) = \sum_{u \in child(v)}(c(u) + m(u)) \tag{7}$$

or as the gate capacitance, $c_g(v)$, of a FF or buffer, if v represents a FF or buffer insertion at n.

The third optimization objective is user defined and is power minimization for LPDP and performance optimization through maximizing the minimum slack at source and sinks for MSDP. *Power consumption* of a partial solution v at node n, denoted $p(v)$, represents the power consumption of the subtree rooted at n under solution v and is defined as in (8) where $p(u)$ is the power consumption of the subtree rooted at the node associated with a child solution of v.

$$p(v) = \frac{1}{2}\left[c_d(v) + \sum_{u \in child(v)} c(u)\right]V_{dd}^2 + \sum_{u \in child(v)} p(u) \tag{8}$$

The performance optimization objective is to maximize the $minimum\ slack$ at the source and sinks, henceforth called margin. The margin at node n under solution v is denoted $t_{slack}(v)$ and is defined as the minimum difference between required arrival time at all k sinks, s_i ($i = 0\ to\ k$,) of T_n and the required arrival time at the output of the first FF, f_i, upstream of each sink.

$$t_{slack}(v) = \min_{i=0\ to\ k}(q(s_i) - q(f_i)) \quad (9)$$

Let T_r be an RC tree with root node r and sink nodes $s_i, 1 \leq i \leq m$. Let the latency constraint at s_i be $f(s_i)$ and the required arrival time constraint at s_i be $q(s_i)$.

$Formulation(LPDP)$: Assign FFs and buffers to the nodes of T_r such that $p(r)$ is minimized, the arrival time at s_i is less than $q(s_i)$ and the latency at s_i is $f(s_i)$.

$Formulation(MSDP)$: Assign FFs and buffers to the nodes of T_r such that the minimum of $\{t_{slack}(s_i), q(r)\}$ is maximized, the slack at each FF is minimized, the arrival time at s_i is less than $q(s_i)$ and the latency at s_i is $f(s_i)$.

3 Algorithm

In this section we describe the details of our algorithm for the LPDP formulation. At each node of T_r, a list of partial solutions for the sub tree rooted at that node is generated by recursively traversing the tree from the bottom up. At a branch node, two child lists of partial solutions must be combined to form a new list of partial solutions. Each entry in the list of partial solutions stores the required arrival time, latency, downstream capacitance and objective function associated with that solution. A partial solution, or *option*, also stores pointers to the options of the child nodes that it is based upon, allowing easy implementation of the top down traversal to trace back the optimal solution. At each node the choice of inserting a FF or buffer is evaluated. The algorithm functions by optimizing three objectives simultaneously. The required arrival time is maximized, the downstream capacitance is minimized and the objective function (either power consumption of the subtree rooted at the current node or minimum margin at the sinks) is optimized. To allow pruning to proceed efficiently, the list can be sorted in terms of one of these parameters. For ease of implementation, we sort in ascending order of required arrival time. The following pseudo code outlines the recursive bottom up propagation of partial solutions. It details the sequence of operations followed by the algorithm. The $FlopInsertion()$ and $BufferInsertion()$ procedures (lines 16 and 18) in the pseudo code simply iterate through the list of options propagated up from child nodes and perform FF and buffer insertion respectively. The resulting lists are then pruned (lines 17 and 19) and combined with the original list (line 20.) This comprehensive list is then passed through the delay and general pruning procedures before itself being propagated up the tree.

```
1  Procedure Insertion(n)
2    if n is leaf node
3      return spec solution;
4    end if;
5    S_m = {};
6    foreach c ∈ child(n)
7      S_c = Insertion(c);
8      S_0 = {};
9      foreach s_0 ∈ S_c
10       s_1 = s_0;
11       q(s_1) = q(s_0) − delay(s_0);
12       push s_1 onto tail of S_0;
13     end for;
14     S_m = Merge(S_m, S_0);
15   end for;
16   S_f = FlopInsertion(S_m);
17   S_f = FlopPrune(S_f);
18   S_b = BufferInsertion(S_m);
19   S_b = BufferPrune(S_b);
20   S_m = S_m ∪ S_f ∪ S_b;
21   S_d = DelayPrune(S_m);
22   S = GeneralPrune(S_d);
23   Return S;
24 end Procedure;
```

3.1 Pruning Rules

Effective pruning is essential to a dynamic programming algorithm. Because the rate at which options are created at each node is super linear, the time complexity of the algorithm would be intractable without pruning. In general, pruning is based upon the comparison of the several optimization objectives, and specifically, determining whether a given solution, s_0, can never result in a more optimal global solution than some other solution, s_1. This determination is made based upon the following property.

Dominance Property
Partial solution s_1 is said to $dominate$ partial solution s_0 if each of the following criteria are true:
a. $f(s_1) = f(s_0)$
b. $q(s_1) \geq q(s_0)$
c. $m(s_1) \leq m(s_0)$
d. $p(s_1) \leq p(s_0)$

If s_1 dominates s_0 then s_0 is not required to find a globally optimal solution and can be pruned. Because the optimality of the algorithm will be compromised by an incorrect pruning condition it is important to prove that the criteria used for pruning is correct.

Proof of Dominance Property
1. Let $\{s_1, s_0\}$ be partial solutions at node n.

2. $f(s_1) = f(s_0)$, $q(s_1) \geq q(s_0)$, $m(s_1) \leq m(s_0)$, $p(s_1) \leq p(s_0)$.
3. Let s_g be any global solution of which s_0 is a partial solution.
4. Substitution of s_1 for s_0 in s_g cannot violate any time constraints or make $p(s_g)$ less optimal.

The Dominance Property is applied in three cases, when a list of flop insertion options is generated, when a list of buffer insertion options is generated and when these lists are combined with the original list. These pruning rules as well as pruning based upon timing constraints are summarized in table 1. Pruning the list of FF insertion options at a given node is performed by the $FlopPrune()$ procedure and is very straightforward. It is a special case of the Dominance property because all such options are known to have identical required arrival times (the end of clock cycle minus setup time) and identical downstream capacitance (the gate capacitance of a FF.) Therefore, the FF with the best objective function for each latency will dominate all other FF insertion options of the same latency.

The $BufferPrune()$ procedure, which performs pruning on the list of buffer insertion options at a given node, while still a special case the the Dominance Property, is complicated by the fact that while they have the same downstream capacitance value (the gate capacitance of a buffer) their arrival times and objective functions may vary. The Dominance Property is applied to buffer insertion options of the same latency to prune suboptimal options.

When lists of options are combined in line 20 of Insertion() options may dominate options from another list. Applying the Dominance Property to all three criteria of optimality is performed by the $GeneralPrune()$ procedure and requires $O(n \log n)$ operations to evaluate the entire list [7].

Options that have a negative required arrival time are ignored, as well as options that have a required arrival time less than the minimum possible arrival time for their latency. In addition, minimum Elmore delay for optimally buffered interconnect is estimated at each node using the closed form optimal buffer insertion length formula as presented in [2] during the initial top down traversal of the tree before bottom up traversal begins. This establishes a hard minimum and is also conservative because it does not take FF setup time into consideration. If the required arrival time of a solution is less than this estimated minimum arrival time the solution cannot result in a viable solution at the root and is pruned by the $DelayPrune()$ procedure.

4 Experimental Results

Parameters for the device and parasitic information have been extracted from some industrial cell libraries and process. For the sake of IP protection, the parameter used is not displayed here. All the nets information presented in this work is not being disclosed to avoid conflicts of IP interest.

Table 1. Pruning Rules

Procedure	Requirements	Pruning Criteria
FlopPrune()	$f(s_0) = f(s_1)$, $m(s_0) = m(s_1)$, $q(s_0) = q(s_1)$	$p(s_0) \geq p(s_1)$
BufferPrune()	$f(s_0) = f(s_1)$, $m(s_0) = m(s_1)$	$q(s_0) \leq q(s_1)$, $p(s_0) \geq p(s_1)$
GeneralPrune()	$f(s_0) = f(s_1)$	$m(s_0) \geq m(s_1)$, $q(s_0) \leq q(s_1)$, $p(s_0) \geq p(s_1)$
DelayPrune()	–	$q(s_0) < q_{min}(s_0)$

Only the general setup information is described here. One type of moderately-sized FF and one type of single-sized buffer are used throughout the experiment. All routing is done on two levels with similar parasitic property. Routing topologies for the nets are obtained from real industrial designs.

4.1 Algorithm Analysis

For a single line, in the worst case with no pruning, the number of options at each node grows exponentially as computation progresses from sink to source. This is because each solution propagated up the tree may give rise to three options, a FF insertion, a buffer insertion, or no insertion. With no pruning our time complexity is therefore $O(c^n)$, where c is 3 in our case. With pruning, however, the number of options at each node is observed to be capped at a level after some initial exponential growth. Such level is roughly proportional to the maximum number of nodes between FFs and insertion points.

The effectiveness of the $GeneralPrune()$ procedure as well as the $DelayPrune()$ procedure was investigated by running the two versions of the algorithm on a very long line, 37mm, with a requirement that six FFs be inserted between source and sink. Figure 2 is a plot of the number of partial solutions at each node of the long line a distance of one unit of decimation apart from sink to source when solved with MSDP. From 2 we can see that the number of options at each node becomes a constant value after an initial period of super linear growth. With $GeneralPrune()$ and $DelayPrune()$, the constant value is brought down by

more than half. This has an effect of roughly 70% reduction in run time due to the reduced complexity in pruning.

In LPDP where both pruning rules have been applied, the general pruning procedure has a dramatic effect on the rate of growth of options at each node. We show in Figure 3 a plot of the number of partial solutions at each node in the main trunk of a test case under the power optimization objective function. The test case has three sinks and a total net length of 9.03mm and was chosen because it is both large and branches more than once, making it a reasonable example of the general case. In the plot, options per node are observed to grow at a nearly constant rate. (Discontinuities are due to merging of branches.) It should be noted that without $GeneralPrune()$ the number of options per node would be equivalent to that of MSDP in Figure 2.

Delay estimation pruning runs in linear time for both estimating delay with respect to the number of nodes in the tree and pruning options at each node with respect to the number of options. The less slack available within a net, the more effective delay estimation pruning becomes, as more solutions fall under such bound.

In conclusion, the runtime of the algorithm depends strongly on the number of options produced at all nodes. The number of options grow exponentially in theory, but effective pruning keeps the number of options under control. Our study shows that the number of options is roughly linearly proportional to the number of nodes in the LPDP case. The most complicated prune, the $GeneralPrune()$, can be performed in $O(n \log n)$ time [7] for each pruning, where n is the number of options at a node. Because options at a node have been empirically shown to be nearly proportional to the number of nodes, applying $GeneralPrune()$ to all nodes will bring the overall runtime complexity to approximately $O(n^2 \log n)$ in practice.

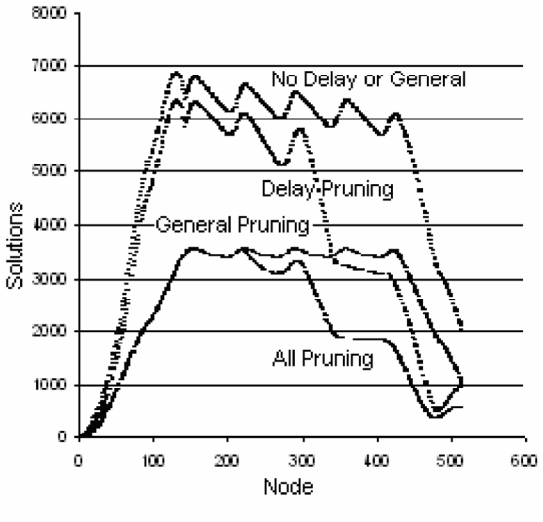

Figure 2. Effects of General Pruning and Delay Estimation on MSDP

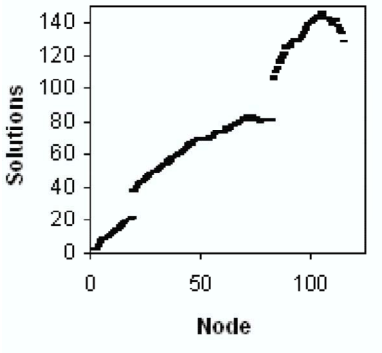

Figure 3. Number of Options in LPDP

4.2 Comparison Between MSDP and LPDP

The effectiveness of the power minimization was demonstrated by comparing the calculated power consumption for 20 real nets in one leading edge industrial design when solved separately under the MSDP and LPDP algorithms. Information about the test cases, as well as the comparison between solution results can be found in Table 2.

The nets are randomly selected from a design cluster. The listed ones are the longer ones (4.5 - 9.8mm) with more sinks (6 - 12). For simplicity, only the maximum latency among the sinks of the net is displayed. As mentioned in the introduction, latency is specified based upon architectural rather than physical considerations.

The margin loss column refers to the factor of decrease in the minimum margin at the source or sinks, which is given by $\frac{|M(MSDP)-M(LPDP)|}{|M(MSDP)|}$. The overall slack refers to the sum of all slack time between the FFs on a net. The overall slack loss is the difference between the overall slack from the MSDP solution and the LPDP solution in terms of percentage of one clock cycle. In all test cases the margin decreases from the MSDP solution to the LPDP solution due to the redistribution of the slack from the sinks and source to the middle. In most examples, LPDP has a reduced overall slack compared to MSDP. The reduced slack is due to fewer repeaters and FFs in the LPDP solution for lower power. Because MSDP has to maximize margin at sinks and may exclude solutions with larger overall delay slack, we observe a few cases where LPDP has more overall slack.

Slack redistribution increases the length of segments between FFs. Thus fewer FFs (eg. one FF driving a fanout of branches instead of multiple FFs each driving one branch) are needed, which in turn reduces the power consumption. FFs are not drastically reduced due to the latency requirement (13.5% on average), but a substantial decrease of buffers (59.7% on average) is seen. The last column shows

Table 2. Experimental Results

Net	Length	Sink	Max Latency	Margin Loss	Overall Slack Loss (%)	$Buffer^*$ (MSDP)LPDP	FF^* (MSDP)LPDP	Power Savings(%)
1	9.77	12	5	7.06	60.8	(25) 10	(14) 10	20
2	7.87	8	5	8.54	0	(21) 6	(10) 9	16
3	8.33	8	6	0.44	20.8	(24) 4	(12) 12	17
4	8.18	6	4	0.03	3.6	(26) 19	(5) 5	6
5	8.24	6	4	0.00	-0.4	(27) 19	(5) 5	7
6	7.17	10	2	0.01	1.2	(22) 8	(7) 7	14
7	8.24	6	4	0.00	-0.4	(27) 19	(5) 5	7
8	6.98	9	2	3.81	11.6	(18) 5	(6) 5	18
9	7.61	9	5	6.27	-34.8	(20) 9	(13) 8	24
10	6.93	6	5	0.18	50	(22) 6	(5) 6	14
11	4.13	6	1	0.34	24	(14) 4	(1) 1	21
12	6.19	6	4	1.12	8.8	(17) 5	(5) 5	15
13	5.30	6	4	0.54	5.6	(17) 5	(5) 5	17
14	9.66	9	5	0.22	17.6	(28) 11	(6) 5	16
15	4.88	6	4	1.14	7.2	(13) 5	(5) 5	12
16	4.51	7	2	0.44	27.2	(14) 5	(3) 3	15
17	9.71	9	5	0.23	18.4	(30) 11	(6) 5	18
18	9.74	9	5	0.24	18.8	(30) 11	(6) 5	18
19	5.25	8	4	6.55	-30.4	(15) 5	(11) 9	19
20	6.06	8	5	11.99	0	(17) 5	(11) 7	28
avg	7.24	8	4	2.46	-	(21) 9	(7) 6	17

the power saving by using LPDP instead of MSDP. As can be seen in net 10, an increase in FFs can result in lower power if it is offset by a reduction in buffers. The combined effect of buffer and FF reduction is that power consumption is reduced by 17% on average and up to 28% for a single net.

5 Conclusions and Future Work

We have shown that significant power reduction in pipelined interconnect can be achieved by properly distributing the delay slack between flip-flops. We have developed an efficient algorithm with pseudo-polynomial time complexity to find flip-flop and buffer insertion solutions that have minimum power and meet the specified latency and delay at multiple sinks of a routing tree. Using a cluster of interconnects in a leading industrial high-performance design, we have shown that without jeopardizing performance, the power of pipelined interconnects can be reduced by 17% for the cluster of interconnects, and the power saving is up to 28% for a single interconnect.

In the future, we will integrate flip-flop and buffer insertion with routing topology generation. We will also consider routing layer assignment and driver/receiver sizing, as well as higher order and RLC interconnect delay models. These will form our overall low power buffer and FF insertion framework that explores power reduction with delay and latency constraints.

References

[1] N. Akkiraju and M. Mohan. Spec based flip-flop and buffer insertion. In *Proc. Int. Conf. on Computer Aided Design*, 2003.

[2] K. Banerjee and A. Mehrotra. Power dissipation issues in interconnect performance optimization for sub-180 nm designs. In *Proceedings of 2002 Symposium on VLSI Circitus*, 2002.

[3] K. Boese, A. Kahng, B. McCoy, and G. Robins. Fidelity and near-optimality of Elmore-based routing constructions. In *Proc. IEEE Int. Conf. on Computer Design*, pages 81–84, 1993.

[4] A. P. Chandrakasan and R. W. Brodersen. *Low power digital CMOS design*. Kluwer Academic Publishers, 1995.

[5] P. Cocchini. Simultaneous insertion of repeaters and flip-flops in high performance vlsi circuits. In *Proc. Int. Conf. on Computer Aided Design*, 2002.

[6] L. He and W. Liao. Full-chip interconnect power estimation and simulation considering concurrent repeater and flip-flip insertion. In *Proc. Int. Conf. on Computer Aided Design*, 2003.

[7] J. Lillis, C. K. Cheng, and T.-T. Lin. Optimal wire sizing and buffer insertion for low power and a generalized delay model. In *Proc. Int. Conf. on Computer Aided Design*, pages 138–143, 1995.

[8] L. P. P. P. van Ginneken. Buffer placement in distributed RC-tree networks for minimal Elmore delay. In *Proc. IEEE Int. Symp. on Circuits and Systems*, pages 865–868, 1990.