Circuit	No. Trans	Clock	Area (sizing)	Area (buffering)	Improvement (%)	Run Time (min)
C_unit	8001	25ns	12.1	12.0	10	15
		20ns	13.7	13.2	19	22
		15ns	22.5	20.2	20	75
		14ns	29.1	25.1	22	89
4-bit ALU	485	10ns	1.46	1.40	10	7

Table 1. Experimental Results

into the template window. A fast version of the TILOS algorithm is then run on the window. The corresponding original circuit cost is then computed from the size of the transistors in the template window and the transistor multiplier factors. The configuration with the lowest cost is identified. If this configuration yields a total circuit cost lower than that of the original, the configuration is chosen for buffer insertion. The sizes from the template including the size of the buffer are then transferred into the original circuit.

5. EXPERIMENTAL RESULTS

Our algorithm is implemented in C and contains an embedded *Pearl* static timing analyzer modified to handle incremental updates. The algorithm has been tested on both combinational circuits and sequential circuits with transparent latches. Results obtained by applying our buffer insertion on two industrial test cases are shown in Table 1. The improvement column of Table 1 represents the relative reduction in size increase to meet timing as compared to the circuit with minimum sized transistors. Both designs in Table 1 use transparent latches and the resulting optimized circuits make heavy use of cycle stealing to lower cost(power). The layout of both circuits are implemented using a transistor level place and route tool (LAS). The netlist and complete parasitics for the optimization of each test cases are extracted from this physical layout of the circuit.

The first test case C_unit is a control block in a microprocessor based design. The second test case 4-bit-ALU is a small arithmetic unit. Five buffers in C_unit and 3 buffers in 4-bit-ALU were inserted by our algorithm. Table 1 compares the active area (total of all transistor gate areas) using sizing alone with the active area achieved using our concurrent sizing and buffer insertion algorithm. Run times are on a Sparc 5 computer with 32MEG of main memory.

Table 1 shows that improvements of 10% to 20% can be obtained and that the improvement increases as the timing constraints become tighter. The existing buffers in the circuit were not removed and both circuits had buffers before optimization. Experiments are being performed to investigate how much further power and area improvements can be achieved by removing all the buffers before performing optimization.

6. CONCLUSION

A method for concurrent sizing and buffer insertion based on area delay cost curves is proposed. The buffer insertion algorithm is incorporated into the TILOS sizing loop At each iteration the method examines potential buffer insertion locations along the critical path. For each promising buffer position, the cost of the buffered circuit for the same delay values of the current TILOS iteration are computed. The decision where to insert buffers is based on these costs. Through buffer insertion the feasible reason of the cost-delay curve is extended to encompass the envelop of area delay curves for all buffer positions.

REFERENCES

[1] J. Fishburn and A. Dunlop, "TILOS: A Posynomial programming approach to transistor sizing," *Proceedings of the IEEE International Conference on CAD* 1985, pp. 326-328.

[2] S. Sapatnekar, V.B. Rao, P. Vaidya, and S.M. Kang, "An exact solution to the transistor sizing problem for CMOS circuits using convex optimization," *IEEE Transactions on Computer-Aided Design, vol. 12*, Nov 1993, pp. 1621-1634.

[3] K.J. Singh and A. Sangiovanni-Vincentelli, "A heuristic algorithm for the fanout problem," *Proceedings of the IEEE Design Automation Conference*, 1988, pp. 357-360.

[4] C.L. Berman, J.L. Carter, and K.L. Day, "The fanout problem: From theory to practice," *Advanced Research in VLSI: Proceedings of the 1989 Decennial Caltech Conference*, 1989, pp. 69-99.

[5] J. Cherry, "Static Timing Analyzer-Pearl," *25th IEEE Design Automation Conference*, 1988.

[6] P. Winston, "Artificial Intelligence 2nd Edition," Addison-Wesley Publishing Company 1984.

estimated from the speedup for the buffered and unbuffered branches, and the sensitivities and slacks for each of the paths using the following formulas:

$$\Delta T(S_{unbuffered}) = R_G(C_{total} - interconnectCap(S_{unbuffered} - C_{buffer} - \sum Gate\_caps(S_{unbuffered}))$$

$$\Delta T(S_{buffered}) = R_G(C_{total} - interconnectCap(S_{unbuffered}) - \sum Gate\_caps(S_{unbuffered})) - buffer\_delay$$

$$\Delta C_{unbuffered} = \sum_i (\Delta T(S_{unbuffered}) - slack_i)/sensitivity_i) + CostOf(buffer)$$

$$\Delta C_{buffered} = \sum_j (\Delta T(S_{buffered}) - slack_j)/sensitivity_j)$$

$$\Delta C(S_{unbuffered}, S_{buffered}) = \Delta C_{unbuffered} + \Delta C_{buffered}$$

It is assumed that the first inverter in the buffer does not get upsized much so C_{buffer} is that of a minimum sized inverter. Also the buffer is assumed to be sized having the same driving capability as that of the original gate. The values of the interconnect capacitances $interconnectCap(S_{buffered})$ and $interconnectCap(S_{unbuffered})$ are obtained using routing estimates and the original layout parasitics. These routing estimates can be determined from information in the layout model described in section 3.1..

Given a net N connecting n objects and occupying a bounding box of length L and width W, the routing capacitance of N is estimated by:

$$Capacitance = k(L+W)\sqrt{n} \qquad with \quad k = constant$$

Let n_0, n_1 and n_2 be the number of objects connected to respectively the original, unbuffered and buffered fanout nets. Also let $(L_0, W_0), (L_1, W_1)$ and (L_2, W_2) be the length and width of respectively the original, unbuffered and buffered nets. The values of L_0, L_1, L_2 and W_1, W_2, W_3 are obtained from the layout model. The layout model maintains and manages the physical location of all transistors in the layout. Except for a small perturbation during buffer insertion these locations are the same as in the original layout. Applying the above formula for the parasitic capacitance to the unbuffered, buffered and original case yields.

$$interconnectCap(S_{unbuffered}) = k(L_1 + W_1)\sqrt{n_1}$$

$$interconnectCap(S_{buffered}) = k(L_2 + W_2)\sqrt{n_2}$$

$$where \quad k = C_{total}/(L_0 + W_0)\sqrt{n_0}$$

4.2. Template Window Acceleration Method

For a given potential buffer insertion location a fast but accurate mechanism is needed for determining whether the proper conditions for buffer insertion are met. The mechanism must also return the proper sizes for transistors after buffer insertion. These are sizes that transistors in the buffered circuit must have so that the delay of the circuit after buffer insertion is that of the original circuit. This operation puts the circuit on the *"sizing and buffering"* curve at delay in Figure 1.

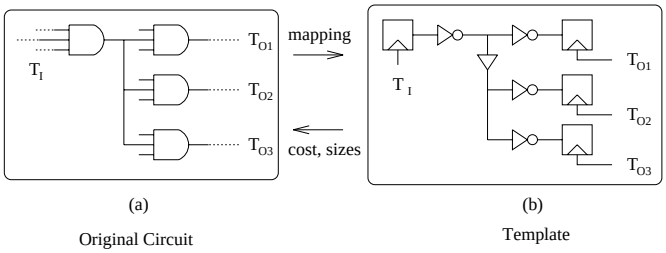

Figure 5. Cost Analysis using a Template Window

To accomplish this efficiently, a model of the buffered circuit in the vicinity of the buffered node is built. To be accurate the template must extend to encompass the entire circuit. In practice a small local neighborhood of the buffered node is modeled using a fixed[1] template similar to the one shown in Figure 5 (b). A buffered version of the original circuit of Figure 5 (a) is modeled by the template of Figure 5 (b).

The timing boundary conditions of Figure 5 (a) are imposed on the template of Figure 5 (b). This is achieved through the use of latches on the periphery of the template. The timing on the clock of the template latches correspond to those of the corresponding ports of Figure 5 (a). The latches are not present in the original circuit and are placed in the template as a mechanism to enforce timing constraints. After sizing, the template window will have the same boundary timing conditions as the circuit of Figure 5 (a).

4.2.1. Increasing Template Accuracy

Although inverters are used to model the gates in Figure 5 (a), due consideration is given to the number of transistors in series in each of the complex gates. To account for the fact that the inverter has only one pullup or pulldown whereas the corresponding gate may have several transistors in series, the inverter transistors are given a multiplier factor for proper calculation of the final cost.

The template window usually stops at the first level of logic after the buffered node. Increasing the template window to include more levels of logic downstream of the fanout node allows the algorithm to produce a more realistic result. Stopping the template window after the first level of logic assumes that only those gates in this first level of logic are sized resulting in a overly pessimistic assumption. The pessimistic assumption causes the cost-delay curve of the template window to rise more sharply than necessary. The cost-delay curve of the template is manipulated to more accurately reflect the real circuit thus making it appear as if the template actually encompassed several levels of logic after the fanout node. By adding capacitance at the output of the fanout inverters and appropriately adding delay to the latch clocks, the cost-delay curve of the template can be made to rise more slowly. At the output of each fanout inverter in the template, an additional capacitor is added and the latch clock signal is delayed appropriately. The capacitor value is computed based on the number of levels of logic to the nearest latch, the capacitance on each of the nodes on that path and the type of gates on the path.

4.2.2. Mapping Configurations to the Template Window

Each configuration proposed by the buffer location generator for each of the nodes in the critical path is plugged

[1] The number of fanouts in the template is not fixed

This involves finding the overall cost of the circuit with the buffer inserted if it were required to meet the timing of the unbuffered circuit at the current TILOS iteration. One way to achieve this is to run the TILOS algorithm from the start on the buffered circuit. To avoid the enormous computational burden of this, TILOS is run on a small local neighborhood of the inserted buffer as described in Section 4.2.. First the local neighborhood of the circuit is copied into the template window. TILOS is then run on the template window. From the results of this local TILOS run the cost of the overall circuit is extrapolated. The buffer insertion part of the overall algorithm is summarized below.

For each node on critical path
  Run buffer location generator function on node
    Add all good locations into candidate location set S_c
For each configuration in S_c
Map configuration onto template window
  Run quick sizing in template window
  Identify best configuration $Best_C$ *from template runs*
  If $Best_C$ *lowers power cost for current timing*
    Add a buffer and update sizes
  Else go to sizing

4.1. Buffer Location Generation

The purpose of the buffer location generator is to produce buffer insertion locations that will be fed into the template window. Given a node N on the critical path the buffer location generator identifies buffer locations that have the potential of reducing circuit cost for the same circuit delay. Each configuration splits the fanouts of N into a buffered set $S_{buffered}$ and an unbuffered set $S_{unbuffered}$. For a node N with n fanouts there are 2^n possible configurations, each one corresponding to a buffer insertion location. Because of the large number of configurations, enumerating all possibilities and rejecting the unsuitable ones is not computationally practical. A *branch and bound* [6] like strategy is developed to reduce the number of configurations that need to be explored. When considering a candidate location the buffer location, generator takes into account both circuit delay and routing capacitance. The routing capacitance translates into cost which is minimized by our algorithm.

The gates in the fanout set of a node N can be categorized into four distinct classes. Each class relates to the criticality of the corresponding path. The four classes of paths are critical path, violating path, near-violating path, and non-violating path. The definitions of the critical and violating paths are in section 2. A path which has the delay slightly smaller than the worst path at the current TILOS iteration is defined as a near-violating path. Thus it is possible for the near-violating path to become a violating path if a buffer is inserted on the path. A path which in all likelihood will never become violating even if a buffer is inserted along the path is defined as a non-violating path. Figure 3 shows the four categories of paths for the fanouts of gate G. By categorizing the paths in this fashion, the good candidate buffer insertion locations become more apparent.

To classify paths into the four categories shown in Figure 3 the slack values on each of the fanout gates of G must be computed prior to buffer location generation. A backward *PERT*-like algorithm from the primary outputs to the primary inputs is used to compute the slack at each of the fanout gates. Using the slack at each fanout, a path can be categorized into the four categories described earlier.

Figure 3 shows four buffer locations (B_1, B_2, B_3 and B_4) that become apparent after classification of the fanouts of gate G. Position B_4 is usually safe in that since no buffer is inserted in any of the violating paths, the violating paths are almost always sped up. However the capacitance of only the non-violating paths is off loaded from the critical path. By moving the buffer to positions B_3 and B_2 additional capacitance is offloaded from the critical path. However the additional delay introduced by the buffer may adversely affect the violating and near-violating paths. Position B_1 is useful only if the increased drive capability of the buffer over gate G offsets the delay introduced by the buffer.

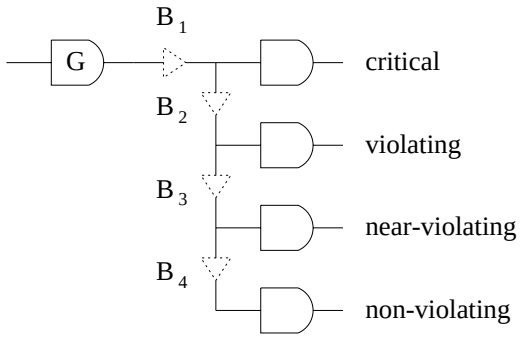

Figure 3. Potential Buffer Locations in Fanouts

Put all fanouts in $S_{buffered}$
$S_{configurations} = \emptyset$
ExplorePossibility($S_{buffered}, \emptyset, \Delta C(S_{buffered}, \emptyset)$)
return($S_{configurations}$)

ExplorePossibility($S_{buffered}, S_{unbuffered}, current\Delta C$)
  If $current\Delta C > 0$ (or some minimum amount)
    $S_{configurations} = S_{configurations} +$
      $(S_{buffered}, S_{unbuffered})$
  For each fanout F_i in $S_{buffered}$
    $new\Delta C = \Delta C(S_{buffered} - F_i, S_{unbuffered} + F_i)$
    if ($current\Delta C < new\Delta C$)
      ExplorePossibility($S_{buffered} - F_i$,
        $S_{unbuffered} + F_i$,
        $new\Delta C$)

Figure 4. Buffer Location Generation Algorithm

The above-mentioned classification of fanout gates and buffer positions avoids considering obviously poor buffer location choices such as buffering critical and non-violating paths but not violating and near-violating paths. The *branch and bound* algorithm described below is inspired from the above observations. Given a buffer insertion location the algorithm uses some of the estimates introduces in sections 2.2.1. and 2.2.2. to estimate the cost difference $\Delta C(bufferedSet, unbufferedSet)$ between the buffered and unbuffered circuit for the current TILOS iteration delay. The algorithm begins by placing all the fanouts in the buffered set $S_{buffered}$. Fanouts are then moved to the unbuffered set $S_{unbuffered}$ only if this improves the cost ΔC. The buffer location generation algorithm is shown in Figure 4. The algorithm performs recursively. After moving a fanout to the unbuffered set procedure *ExplorePossibility* calls itself recursively to further explore the ramifications of that move.

The cost function ΔC for a given configuration can be

2.2.2. Type B Buffer Insertion

Figure 2(b) illustrates the second kind of buffer insertion called *Type B* buffer insertion. Here the gate G_B is driving a number of fanout loads, some on violating paths and some on non-violating paths. In this case it is beneficial to add a buffer on the non-violating paths. This isolates the violating paths from the slowdown due to the load C_k of non-violating fanouts. Using a simple RC model the speedup of the violating path in Figure 2(b) after buffer insertion is $\Delta T_B^j = R_B \cdot (C_j + C_k) - R'_B \cdot (c + C_j)$. The speedup for the non-violating path is $\Delta T_B^k = R_B \cdot (C_j + C_k) - R'_B \cdot (c + C_j) - r \cdot C_k$.

An approximate estimate of the cost(power) reduction due to buffer insertion can also be derived for type B buffer insertion. This estimate is used to quickly identify promising type B buffer locations. A more accurate method described later in this paper is then applied on these promising locations to get a good estimate of the actual cost reduction.

3. OVERALL ALGORITHM

The buffer insertion algorithm is incorporated into the TILOS algorithm main loop. TILOS starts with transistors in their minimum sized configuration. With minimum transistor size the circuit delay is normally larger than the clock specification and latches in the circuit give rise to timing violations. In order to verify and update timing information, the static timing analysis tool **Pearl** [5] is chosen as our main timing engine. The static timing analyzer is modified to efficiently handle incremental timing updates. When a circuit element is changed, the effects of timing downstream of its cone of influence are incrementally updated. Because of convergent fanin and latches, the effects of a change usually die out after a few levels of logic.

Also, an efficient priority queue heap data structure for all timing constraints is maintained in our timing analysis engine. The top of the heap contains the most violated constraint. After a change to the circuit, the timing is incrementally recomputed, all affected constraints are examined and the heap is adjusted with new most violating constraint at the top of the heap. This entire operation is very efficient and a number of complete updates per second can be performed even for circuits of several thousand transistors.

To perform buffer insertion the TILOS main loop is augmented to first sweep through the critical path for buffer insertion. Buffer positions for which cost can be reduced while maintaining the timing of the current iteration on all circuit paths (same violations and slacks on all latch inputs) are considered and the *best* such location is identified. If such a location exists then the cost-delay curve for this buffer position is below the cost-delay curve of sizing alone for the current delay (see Figure 1). The best buffer location will have a low cost and a small cost slope for the current circuit delay. As mentioned earlier it is assumed (and is generally the case) that the buffer curve will remain below the sizing curve for smaller circuit delays. Without this assumption it could become necessary to remove buffers as the circuit speed is increased during optimization.

If a suitable buffer position is found, the buffer is inserted, the neighborhood of the buffer is resized to maintain the same circuit delay and sizing is skipped for the current TILOS iteration. Otherwise the critical path is scanned again to determine the best transistor to size according to the TILOS algorithm. After the circuit has been modified either through buffer insertion or sizing, timing on all paths is incrementally recomputed and the constraint heap readjusted.

An overview of the overall algorithm is given below. Details of how the buffer insertion position is found are given in the next section.

```
Downsize transistors to minimum size or cost
While (slack(WorstConstraint) < 0)
  For each node in critical path
    See if a buffer can help reduce cost
  If there is a good buffer position
    Insert buffer at best position
    Resize buffer neighborhood for same delay
  else
  For each transistor on critical path
    Find improvement by sizing transistor
  Increase size of best transistor
  Update timing and critical path information
```

3.1. Layout Model

In most existing circuit optimization methodologies that take into account layout parasitics, optimization and physical layout generation operate as two loosely coupled alternating steps. Optimization is performed after an initial estimate of placement is known. The placement is recomputed after optimization and the loop is repeated until all constraints are met. For high performance design it is beneficial to couple the two steps more closely so that updated layout information is available at all times to the optimizer. In our approach a model for layout is maintained and kept updated as the optimization progresses. Through the use of this model the optimizer capitalizes on available layout area and steers the optimization process to preferentially upsize those devices as well as insert buffers with the least impact on area. Since the circuit with the smallest active area is not necessarily the one with the smallest layout area, the optimizer under user control can be adjusted to favor a reduction in active area, layout area, or a combination of both.

During sensitivity calculation the cost of increasing a transistor includes the physical area. To estimate the physical area increase due to a transistor size increase the x and y coordinates of transistors with their widths and lengths are stored in a file called *layout model*. By utilizing the layout model which contains not only the locations of transistors but also the slacks in layout area, both transistor sizing and buffer insertion can be performed with the least possible layout area. The details of buffer insertion algorithm are described next.

4. BUFFER INSERTION

The previous section described how the buffer insertion algorithm resides within the overall sizing and buffer insertion strategy. This section describes how the best buffer position (if one exists) is selected. The buffer insertion algorithm is broken up into two parts.

In the first part buffer insertion positions are generated. Estimates for the buffer insertion cost developed in sections 2.2.1. and 2.2.2. are used to quickly prune out buffer insertion positions that have little hope of yielding good results. This part of the algorithm described in Section 4.1. can be tuned to generate only a few possibilities thus speeding up run time or generate a more exhaustive list thus improving the final result at the expense of run time.

In the second part each promising buffer location generated above is examined more carefully to accurately determine the overall cost of inserting a buffer at that location.

buffer curve dips below the sizing curve at T_i, the curves almost never intersect for $T < T_i$. Therefore when a buffer is inserted at T_i, it is never removed.

Our buffer insertion method is split into two parts. The first part is the buffer location generator which proposes a buffer insertion location. The second part is an *oracle-like* mechanism that returns the lowest possible cost value for that buffer location at the current delay. The solution with the lowest power cost is chosen. In theory, for the optimum result of all possible buffer insertion locations must be considered at each TILOS iteration. Also in theory, to properly compute the lowest possible cost value for a given buffer location TILOS must be run again on the entire circuit. Because of the prohibitively large compute times associated with each of these processes, we introduce algorithms that solve these problems approximately and still yield good results.

To quickly determine a set of promising buffer insertion locations, a *branch-and-bound* like algorithm which examines nodes on the current critical path is introduced. The algorithm can be tuned to enumerate only the most promising locations or a greater number of candidate solutions allowing the user to trade off optimality with run time.

Given a buffer location, to quickly determine an accurate estimate for the lowest cost for the current delay a *Template Window Acceleration Method* is introduced. The window models the behavior of the circuit in the vicinity of the buffer. Instead of computing the new sizes for the entire circuit, sizing is performed on this window. The cost for the entire circuit is then extrapolated from these sizes. In theory, for optimal results, the size of the window must encompass the entire circuit. In practice good results are obtained using a window a few gates upstream and a few gates downstream of the buffered node.

In section 2 the fanout problem in sizing is described. The overall algorithm of concurrent transistor sizing and buffer insertion is described in section 3 followed by the *Template Window Acceleration Method* in section 4. The experimental results are discussed in section 5 and finally the conclusion is in section 6.

2. FANOUT PROBLEM IN TRANSISTOR SIZING

Because of the fanout problems [3,4], a gate on the critical path with large load capacitance may get excessively sized as the result of transistor sizing. Since power dissipation is roughly proportional to the capacitance of the circuit, large transistors consume more power and may increase total chip area. Buffer insertion is used to reduce the amount by which such transistors may need to be sized in one of two ways. First the drive capability of a gate may be increased by adding a buffer at its output. This is called type A buffer insertion. Alternatively the drive requirement of the gate can be reduced by off loading from the output node, the fanout gates that are not on the critical path. This is called type B buffer insertion. Both type A and type B buffer insertion are described in more detail in the section.

2.1. Definitions and Terminologies

A transistor T has a width and a length. Only the transistor width is changed during transistor sizing. The transistor area is $width * length$. A path $P = \{f_0, G_0, f_1, G_1, ..., G_{m-1}, f_m\}$ in the circuit network is an alternating sequence of nodes and gates. If outputs of the circuit are latched and a path P violates setup constraint, the path P is defined as a **violating path**. The path which never violates setup constraints under the given timing constraint is also defined as a **non-violating path**.

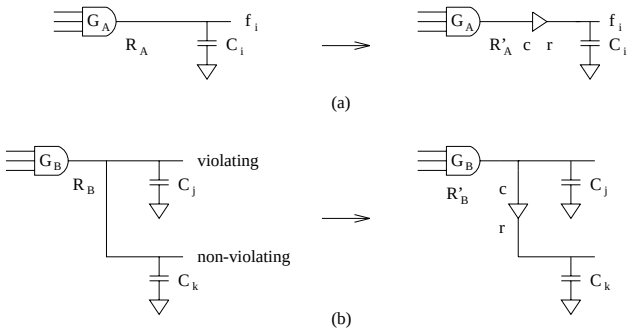

Figure 2. Buffer Insertion (a) Type A (b) Type B

Definition 1 *The* **critical path** *is the most violating path.*

Definition 2 *The* **transistor sensitivity** *is the improvement in the delay of the critical path per unit increase of area* $(\Delta T/\Delta A)$.

2.2. Basic Concepts

Concurrent sizing and buffer insertion considers the tradeoffs between sizing and buffering. During each sizing iteration of the TILOS algorithm, a transistor in the critical path is selected to be upsized depending on the *transistor sensitivity*. As shown in Figure 1 the slope of the sizing curve becomes high when the circuit delay approaches the minimum achievable delay T_{min}. This large slope trajectory means low *transistor sensitivity*and hence requires large size increase ΔW to achieve a given delay improvement.

For the fanout problem there are two cases for which buffer insertion is advantageous. The buffer insertion case of Figure 2(a) is referred to as *Type A* buffer insertion. A buffer is added before the fanout to increase the drive capability of the fanout gate G_A. Alternatively a buffer can be added to the non violating path in Figure 2(b) to reduce the load on gate G_B thus speeding up the violating path. This is referred to as *Type B* buffer insertion.

2.2.1. Type A Buffer Insertion

In Figure 2(a), the gate G_A is driving a large capacitance C_i. Assuming a simple RC delay model the initial delay before buffer insertion is computed as $T_A = R_A \cdot C_i$. After buffer insertion the delay becomes $T'_A = R_A \cdot c + r \cdot C_i$ where r is the resistance and c is the capacitance of the inserted buffer (for explanatory simplicity treating the buffer as if it were a single inverter gate). The speedup of the circuit due to buffer insertion is then $\Delta T_A = T_A - T'_A$.

Our method requires an estimate of the cost (power) consumed by the buffered circuit for the same delay as the unbuffered circuit. In the buffered circuit gate G_A can be downsized because it is now driving a much smaller load. If the new downsized gate G_A has resistance R' the delay of the buffered circuit becomes $R' \cdot c + r \cdot C_i$. If S_A is the sensitivity of gate A then a rough estimate for the area gain is $\Delta A = \Delta T_A / S_A - A_B$ where A_B is the area of the buffer. The above method is used to quickly identify promising type A buffer locations. On each promising location a more accurate method for estimating the cost is used for final validation. This method is described later in this paper.

CONCURRENT TRANSISTOR SIZING AND BUFFER INSERTION BY CONSIDERING COST-DELAY TRADEOFFS

Juho Kim[1] *Cyrus Bamji* [2] *Yanbin Jiang*[3] *Sachin Sapatnekar*[4]

[1]Sogang University, jhkim@ccs.sogang.ac.kr
[2]Cadence Design Systems, cyrus@cadence.com
[3]Iowa State University, ybjiang@iastate.edu
[4]Iowa State University, sachin@iastate.edu

ABSTRACT

A method for concurrent transistor sizing and buffer insertion is proposed. The method considers the tradeoff between upsizing transistors and inserting buffers and chooses the solution with the lowest possible power and area cost. The method operates by analyzing the feasible region of the cost-delay curves of the unbuffered and buffered circuits. As such the feasible region of circuits optimized by our method is extended to encompass the envelop of cost-delay curves which represent the union of the feasible regions of all buffered and unbuffered versions of the circuit. The method is efficient and tunable in that optimality can be traded for compute time and the method can in theory produce near optimal results.

1. INTRODUCTION

With the growing demand for portable electronic systems, low power with high speed has become a major design consideration along with area. Optimization for low power can be applied at various stages of the design process from the architectural level to the physical layout implementation of the circuit. Gate/transistor sizing is one of the well-known methodologies for timing optimization at the netlist level.

For a given circuit topology there is a cost-delay tradeoff curve [1,2] shown by the sizing curve in Figure 1. Each point on the figure represents a particular power (size) and delay configuration of the circuit. The sized circuits above the curve are suboptimal in that the same timing can be achieved at a lower cost. The region below the curve is infeasible and no purely sized circuit can have a delay and cost in this region. The sizing curve in Figure 1 shows that when the delay is small, further improvements in delay reduction come at a high cost increase.

The TILOS algorithm [1] is used to size transistors in a circuit to produce a sized circuit near the optimal cost-delay curve. TILOS begins by downsizing transistors to their minimum size. TILOS then begin a series of iterations during which transistors are selectively upsized to make the circuit meet timing. During this process the circuit sweeps the cost-delay curve shown in Figure 1(*sizing* curve) from right to left.

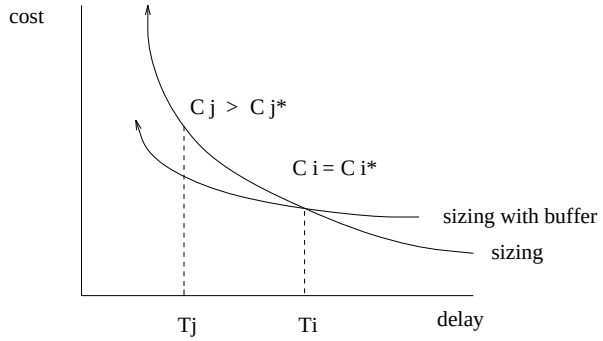

Figure 1. Cost vs. Delay Curve; sizing and buffering (C_j* = cost with buffer insertion at delay T_j)

For a given delay requirement, through careful buffer insertion it is possible to obtain a circuit with less overall cost(power) than is achievable through sizing alone. Also, through buffer insertion it is possible to improve circuit delay beyond what is possible through sizing alone. Adding a buffer changes the topology of the circuit resulting in the new cost-delay curve shown by the *sizing with buffer* curve in Figure 1. Each possible different buffering of the circuit gives rise to a different curve. By adding buffers so as to be on the curve with the lowest cost for a given delay T_j, an optimal sizing-buffering solution can be reached.

Unlike existing methods which perform buffer insertion and sizing in distinct phases, our method concurrently performs buffer insertion along with sizing. Trade-offs between sizing and buffering are directly considered by our method. During the TILOS algorithm the sizing curve is swept from right to left. At some point the delay T_i in Figure 1 is reached. At this point it is beneficial to add a buffer in such a manner that optimization now proceeds on the lower *sizing with buffer* curve. Our method detects this condition in the vicinity of delay T_i and adds a buffer at an appropriate location. After buffer insertion at delay T_i, the circuit follows the *sizing with buffer* curve which is lower than that for sizing alone. As optimization progresses other buffers will be added and the optimization sweeps an envelop of curves representing the boundary of circuit speeds achievable by sizing and buffer insertion.

Strong conditions that guarantee that the buffer with sizing curve once below the sizing curve will remain below the sizing curve can be derived but are beyond the scope of this paper. The conditions under which buffers are inserted in our method do not always satisfy these conditions entirely. In practice however been empirically observed that once the