Path Based Buffer Insertion[*]

C. N. Sze, Charles J. Alpert[†], Jiang Hu and Weiping Shi
EE Dept., Texas A&M Univ., College Station, TX 77843; {cnsze,jianghu,wshi}@ee.tamu.edu
[†] IBM Corp., 11501 Burnet Road, Austin, TX 78758, USA; alpert@us.ibm.com

ABSTRACT

Along with the progress of VLSI technology, buffer insertion plays an increasingly critical role on affecting circuit design and performance. Traditional buffer insertion algorithms are mostly net based and therefore often result in sub-optimal delay or unnecessary buffer expense due to the lack of global view. In this paper, we propose a novel path based buffer insertion scheme which can overcome the weakness of the net based approaches. We also discuss some potential difficulties of the path based buffer insertion approach and propose solutions to them. A fast estimation on buffered delay is employed to improve the solution quality. Gate sizing is also considered at the same time. Experimental results show that our method can efficiently reduce buffer/gate cost significantly (by 71% on average) when compared to traditional net based approaches. To the best of our knowledge, this is the first work on path based buffer insertion and simultaneous gate sizing.

Categories and Subject Descriptors

B.7.2 [**Integrated Circuits**]: Design Aids - Placement and Routing; J.6 [**Computer-aided Engineering**]: Computer-aided Design

General Terms

Algorithms, Performance, Design

Keywords

Buffer Insertion, Interconnect Synthesis, Power Minimization, Global Routing, Layout, Physical Design

1. INTRODUCTION

Buffer insertion is widely recognized as an essential technique for interconnect optimization [7] while interconnect is a fundamental limit [10] for VLSI technology progress. The importance of buffer insertion has resulted in numerous algorithmic and methodologic works. Perhaps the most influential work is the classic van Ginneken's algorithm [12]. Given a Steiner tree spanning a signal net and candidate buffer locations on the tree, van Ginneken's dynamic programming (VGDP) algorithm can find the maximum timing slack solution optimally in quadratic time. This algorithm is extended to handle buffer cost and buffer library in [18]. A wire segmenting technique is suggested in [2] for generating candidate buffer locations. The noise avoidance issue is addressed in buffer insertion in [3]. Higher order delay models are adopted in buffer insertion in [4]. For 2-pin nets, quadratic programming based approach [8] and closed form buffering solutions are proposed in [2, 9, 11]. Recently, an $O(n \log n)$ buffer insertion algorithm is developed [23].

[*]This work is partially supported by Semiconductor Research Corporation under contract number 2003-TJ-1124, the DAC Graduate Scholarship, and NSF grants CCR-0098329, CCR-0113668, and EIA-0223785.

Recently, an industry study [22] predicts that 35% of the cells on a chip will be buffers at 65*nm*. The huge number of buffers may affect various aspects of circuit design and performance including timing [7], power dissipation [18], signal integrity [3], placement and routing congestion [5, 22]. Therefore, buffer insertion needs to be conducted in a more elaborated manner to push the envelope of performance.

In fact, most of the previous works on buffer insertion are net based, i.e., buffer insertion is performed on one net after another individually. Even though the buffer insertion problem with net based formulations is relatively easy to solve, it may lead to sub-optimal path delay or unnecessary buffer usage due to the lack of global view. The weakness of net based buffer insertion can be illustrated through a very simple example in Figure 1. Consider two nets A and B along a critical path in the circuit. If we perform buffer insertion on net B first, 4 buffers are needed on B and then no buffer is needed on A for satisfying the critical path timing constraint. This is denoted by solution $S1$. However, the constraint can also be satisfied by putting 1 buffer on both A and B, denoted as $S2$. Optimizing the entire path may get a better solution and certainly can lead to a better deployment of buffering resources. Usually, net based dynamic programming algorithms such as [18] do not have a global view, nets which are processed first tend to over-consume buffer resources.

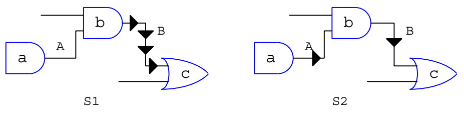


Figure 1: Net based buffer insertion solutions depend on net ordering.

Despite their current popularity, the net based buffer insertion methods, without global view of the whole combinational circuit, will become inadequate for future technologies. In [19, 20], network based buffer insertion algorithms are proposed. In these approaches, buffer insertion is performed on all nets between PI/registers and registers/PO simultaneously through Lagrangian relaxation. However, both works include a restrictive assumption that buffers are inserted at every branch node to simplify the calculation of delay. In practice, whether or not buffers are necessary at certain branch node depends on timing constraints of related paths. Due to path re-convergence, it is very difficult to perform network based buffer insertion without the assumption. Although their works usually produces good results, they do not scale very well. Indeed, in [20], the CPU time consumption explodes for the larger testcases. When buffer insertion is considered beyond the limit of nets or gates, it is natural to consider gate sizing at the same time. In [14], a greedy heuristic on integrated gate sizing and buffer insertion is proposed. However, it neglects wire delays. The network based methods make severe oversimplifications in order to achieve a solution, such as ignoring wire delays entirely. Our method takes advantage of some global optimization without sacrificing any of the modeling accuracy that the net based approaches provide.

In this paper, we propose a path based buffer insertion heuristic in order to minimize buffer/gate cost subject to path timing constraints. This approach is in the middle between net based and network based methods. However, it can achieve both better solution quality and faster computation runtime. Since our path based approach can easily handle false paths, the solution quality can be much better than network based methods. Besides, instead of relying solely on static timing analysis, a fast estimation on buffered delay is applied on the entire network so that a better global view is obtained. Our path based buffer insertion algorithm is based on the VGDP algorithm since it is robust and sophisticated enough to handle different instances. However, directly using VGDP may induce problems and we successfully solve those problems by a set of techniques such as off-path required arrival time estimation and gate sizing at sinks. Experimental results show that the usage in buffer/gate cost is reduced by 71% on average through our approach compared with traditional net based algorithms. The runtime is also reasonably fast.

2. PROBLEM FORMULATION

A placed and routed circuit can be formulated as a directed acyclic graph (DAG) $G = (V,E)$. An example is shown in Figure 2. A vertex $v \in V$ can be either (1) a primary input (PI) / primary output (PO) (e.g., nodes a and r in Figure 2(b)), or (2) a pin of a module (e.g., f and i), or (3) a Steiner node on the route (e.g., e, shown as double circle in Figure 2(b)), or (4) a candidate buffer insertion location (not shown in Figure 2(b)). An edge is either an interconnect wire (solid line) or an input-to-output path (dotted line) within a module.

In this paper, interconnect wires are modeled as a distributed RC network and we adopt the Elmore delay model. Thus, an interconnect w is annotated by the corresponding resistance $R(w)$ and capacitance $C(w)$. A buffer b_i in the buffer library is defined by its load capacitance $C(b_i)$, intrinsic delay $T(b_i)$ and output resistance $R(b_i)$, while $W(b_i)$ represents the buffer cost which can be either buffer area or power.

A module is identified by the delay $T_{j,i}$ for each input-to-output path, the capacitance C_j of the input pin and the resistance R_i of the output pin. If x_i is the size of the module, $C_j = \hat{C}_j x_i + f_j$ and $R_i = \hat{R}_i / x_i$, where $\hat{C}_j, \hat{R}_i, f_j$ are the unit size output resistance, unit size gate area capacitance and gate perimeter capacitance of the module. In this paper, the size of each gate is selected from the set $S = \{x_1, ..., x_n\}$. Each primary input is annotated with a user-defined arrival time while each primary output a required arrival time.

There may exist buffer blockages in the floorplan, for example, the shaded box in Figure 2(a). When a buffer insertion candidate location overlaps with the region of the buffer blockages, it is restricted such that no buffer can be placed. In other words, there exists no buffer candidate location within the buffer blockage region.

Signal slew rate is also considered in our path based buffer insertion algorithm. For a signal propagating along a wire, we employ a simple metric of $propagation\_slew = \ln 9 \cdot Elmore\_delay$ [7]. The slew rate at the receiving end of a wire depends on both the propagation slew rate and the launching slew rate at the driving end of the wire and is given in [13] by

$receiving\_slew = \sqrt{launch\_slew^2 + propagation\_slew^2}$.

In our buffer insertion algorithm, any buffer solution with *receiving_slew* greater than a certain threshold will be discarded.

The problem of **simultaneous gate sizing and buffer insertion** is defined as follows. Given a DAG which represents a placed and routed combinational circuit, a buffer library, a set of buffer candidate locations, a set of buffer blockage regions, find a buffering solution such that the overall cost of buffers and gate sizes is minimized. Buffering solution is in terms of the locations and types of buffers inserted, and sizes of the gates. At the same time, the solution is subject to the constraint of both the arrival time at each primary input and the required arrival time at each primary output as well as the slew rate requirement.

As mentioned in Section 1, van Ginneken's dynamic programming (VGDP) approach is very flexible and efficient so that it can be easily applied to buffer blockage avoidance (by selectively setting candidate buffer locations) while considering buffer cost (i.e., area/power), buffer polarity and slew rate [17]. In order to utilize the flexibility and efficiency of VGDP, we intend to use net based buffer insertion algorithm as a building block for path based buffer insertion. In this way, we abstract the routing tree of the circuit and ignore all the details (i.e., Steiner node and interconnect tree structure, etc.) within the routing tree. An example of our circuit model is shown in Figure 2(c). In our model, we abstract all interconnect routing so that vertices only represent PI/PO of the circuit and input/output pins of modules (we use this definition for vertex hereafter in this paper) while edges only for input-to-output paths within a module. The routing tree is identified by its root vertex. For example, the routing tree $RT(c)$ is rooted at vertex c and with sinks k and m. In a combinational circuit, the root of a routing tree is either a PI vertex or an output pin of a module, while a sink is either a PO vertex or an input pin of a module.

3. NET BASED BUFFER INSERTION

In this paper, we assume an efficient VGDP algorithm is given such that it considers buffer blockage, buffer polarity, buffer area/power and slew rate. For a routing tree, if the required arrival time (RAT) at each sink vertex is given, VGDP algorithm traverses every candidate buffer location v_i of the tree in a bottom-up manner, propagating a set of solutions in the form of (c,q,w) which stands for downstream load capacitance, RAT, and total buffer cost respectively. Each solution reflects the intermediate results of a buffering solution on the subtree rooted at v_i. When the propagation reaches the driver (i.e., root vertex), a set of solution with different cost-RAT tradeoff is obtained. If the arrival time (AT) at the root vertex is given, we pick the solution with minimum buffer cost while timing requirement is satisfied.

Conventionally, net based buffer insertion for the whole circuit is accomplished by iteratively performing the following steps:

1. Static timing analysis (STA) and obtain the RAT and AT at every vertex

2. Perform buffer insertion for a routing tree with the AT and RAT obtained from STA (according to a specific net order. Discussion of net order is in Section 6.)

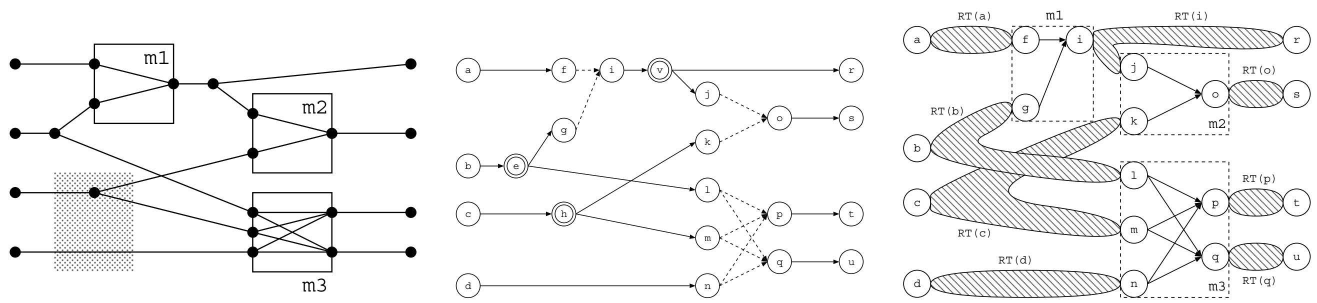
3. Update STA results (of the fanin/fanout cone of the buffered routing tree) and perform buffer insertion for the next routing tree

However, this is in fact a greedy algorithm and in our experiment, we found that the buffering solution of this approach is far from optimal even we let the iteration run unboundedly in order to refine the buffering solution for buffer cost reduction. Our observation to the problems of net based buffering include:

- STA is usually not buffer aware so the timing estimation at the first iterations are inaccurate and the circuit is very timing critical. Hence, the first processed nets tend to over-consume buffer resources (the case of net B in $S1$ of Figure 1).

- Since the algorithm is in nature greedy and a poor buffering decision to a routing tree $RT(a)$ due to incorrect timing estimation may lead to a poor buffering decision at other routing trees $RT(b)$ where b is a transitive fanin/fanout of a. Thus, earlier buffering decision may have degraded the quality of the whole buffering process, which cannot be improved in latter iterations. This is especially true for those nets along a critical path from PI to PO.

- Buffering solution of a routing tree is highly depending on the criticality of the sinks, which in turn depends on buffering of



(a) An example of a combinational circuit

(b) The corresponding DAG

(c) The circuit model with routing tree abstraction

Figure 2: Example of combinational circuit models

their fanout routing tree. Therefore, the criticality can be substantially different from the results of STA due to buffer blockages, which brings significant error in final buffering solution.

4. PATH BASED BUFFER INSERTION

In this section, we propose our path based buffer insertion algorithm (PBBI) as described in this section. The key elements of our buffer insertion algorithm are (1) buffer aware static timing analysis, (2) path based VGDP buffer insertion, and (3) off-path required arrival time estimation.

Our algorithm starts with a buffer aware static timing analysis which also takes care of buffer blockages. In such an approach, the resultant AT and RAT is comparatively much more accurate than ordinary STA and we will not over-consume buffer resources even at the very beginning. After that, a list of k most critical paths is obtained accordingly. Then, VGDP is applied to the paths. In this approach, the root and the sink along the path have a fixed AT and RAT respectively, which in turn produces a relatively good buffering solution. For all other sinks (namely the off-path sinks), we propose an approach to adjust their RAT values to become more accurate and then the RAT are fed into the VGDP algorithm.

4.1 Buffer Aware Static Timing Analysis

Critical path method is widely used as a tool for static timing analysis (STA) [21]. It propagates the static delay information throughout the circuit. However, the delay along interconnect changes during the process buffer insertion which is a main source of error of net based buffer insertion algorithms, as mentioned in Section 1. A work which predicts the post-buffering delay is in [6]. The work derives delay equations along a buffered wire segment considering buffer blockages and applies the equations for delay estimation upon multi-pin nets. Experimental results show that the delay estimation merely produces insignificant errors. Towards our problem, we verify in our experiments that integrating this buffer aware delay estimation with STA provides a good guide for buffer insertion using VGDP. With buffer aware STA, early buffering step will not over-consume buffer resource which trap the overall solution into a local optimum.

Buffer aware STA not only provides a good basis for VGDP algorithm on a path, but it is also a crucial element of the whole path based buffer insertion algorithm. For example, if we are performing buffer insertion on a critical path p_c from PI to PO, it propagates a set of solutions from the PO vertex with accurate RAT information (since the RAT at PO is fixed by user specification). During the propagation, there may exist some branch paths such that the RAT of those paths is needed to compute the solution sets. In such a way, if the RAT of branch paths is not accurate since STA is not aware of buffering along those branches, VGDP may think that the branches are more critical than p_c, which destroys the solution quality of the path based buffer insertion algorithm.

4.2 Path Based Buffer Insertion

In order to accomplish path based buffer insertion, a list of distinct paths must be first obtained. With the help of buffer aware STA, k most critical paths can be found using a polynomial-time algorithm in [15] (k can be changed during the progress of the algorithm). The parameter k is a tradeoff between quality and speed while k can be determined on the fly - stop finding the next critical path if the slack of the path is less than a specific value. Note that the accuracy of our algorithm can also be improved when false paths are detected and only sensitizable critical path are selected. In the whole circuit, each routing tree have to be processed once. Therefore, if the list of paths are overlapping with each other, we delete the common vertices from the less-critical path and cut it into different distinct paths. For example, if the 3 most critical paths in Figure 2(c) is $\{b,g,i,r\};\{a,f,i,j,o,s\};\{c,k,o,s\}$, after removing common vertices, the list of distinct paths becomes $\{b,g,i,r\};\{a,f\};\{j,o,s\};\{c,k\}$.

After getting a list of distinct paths, for each path, the algorithm treats all routing trees along the path as one big routing tree. The merging process is simple since the routing trees is cascaded together such that the sink (an input pin of a module) along the path merges into the root (an output pin of the same module) of the fanout routing tree. The merged vertex is treated as a candidate buffer location such that a special buffer must be inserted. The parameters of the special buffer corresponds to the capacitance/delay/resistance of the pin-to-pin path of the module. In Figure 2(c), the merged routing tree on the path $\{b,g,i,r\}$ consists of three sinks l,j,r rooted at b. g and i are merged into one buffer location with a special buffer b_s according to input-to-output path of $M1$ (The delay model of a buffer is similar to that of an input-to-output path of a module). After all, VGDP algorithm is applied to the merged routing tree. After all paths have been processed, there may exist some routing trees in which no buffer insertion has been performed. They are all comparatively non-critical and net based buffer insertion can be carried out for each of those nets.

4.3 Off-Path Required Arrival Time Estimation

Since the PBBI algorithm performs buffer insertion in a path-by-path basis, the buffering solution may violate the timing constraints. An example is shown in Figure 3. The straight line represents input/output path within a module and dotted curly line stands for a path. Assume that a is a PI vertex and z is a PO vertex and the algorithm processes the path $p_1 = \{a \rightsquigarrow g \rightarrow h \rightsquigarrow p \rightarrow r \rightsquigarrow z\}$ prior to another path $p_2 = \{i \rightsquigarrow q\}$. When applying VGDP on the path p_1, actual buffering solution may reduce the delay along $g \rightarrow h \rightsquigarrow p$ to a value which is less than our delay estimation using buffer aware STA. Since the delay between a and z is bounded above by the an user specified RAT and AT, the delay along the paths $a \rightsquigarrow g$ and $r \rightsquigarrow z$ could be larger than our estimation. In such case, it could happen that even with the minimum-delay buffering along the path $i \rightsquigarrow q$, the delay along $p_3 = \{a \rightsquigarrow g \to i \rightsquigarrow q \to r \rightsquigarrow z\}$ still violates the timing constraint. After each time we perform path based buffering insertion along a path, the AT and RAT of fanin and fanout cone for each vertex of the path are updated, so the only situation which causes the problem is that there exists re-convergence along the path we are performing buffer insertion. As in the previous example, we are inserting buffer along the path p_1, assuming that there exists a path $i \rightsquigarrow q$ in the list of distinct paths. The required arrival time of i (denoted as RAT_i) does not reflect the buffering solution along $r \rightsquigarrow z$ since the final buffering solution is unknown until the whole path p_1 is done. In this consideration, an equation for spreading out the slack of a path to all the routing tree is needed, which is presented in Theorem 1. In the following, we first derive the equation and revisit the problem in the example at the end of this section.

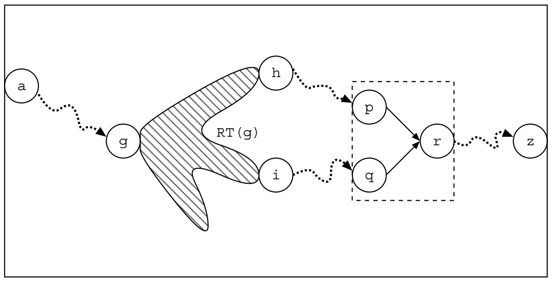


Figure 3: A problem of path based buffer insertion algorithm

Considering a circuit which only contains a cascade of two-pin routing trees $\{RT(s_1), RT(s_2), \ldots, RT(s_k)\}$ as shown in Figure 4. For each routing tree $\{RT(s_i)\}$ with root s_i, the only sink is t_i. With optimal buffering, we can find the minimum delay $\underline{d_i}$ for each $RT(s_i)$. And based on the minimum delay, for each s_i, AT_{s_i} and RAT_{s_i} is propagated from PI and PO respectively according to the user specified timing constraint. Note that since the circuit is a sequence of 2-pin net, the slack $RAT_{s_i} - AT_{s_i}$ must be the same for all s_i. If $AT_{s_i} = RAT_{s_i}$, the circuit has zero slack and only the minimum-delay buffering solution can fulfill the timing constraint. However, if slack> 0, different buffering with smaller buffer cost is possible. We can denote $RAT_{s_i} - AT_{s_i}$ as "useful slack" resource such that buffering algorithm uses it to reduce the total buffer cost. Intuitively, since the ratio of timing improvement to buffer cost is usually greatest around the middle region of the cost-delay tradeoff curve, a buffering solution for the whole circuit with minimum cost tends to spread the "useful slack" to each $RT(s_i)$. Based on a buffering solution B with minimum cost, we can calculate the delay d_i^B , $AT_{s_i}^B$ and $RAT_{s_i}^B$ for each $RT(s_i)$ accordingly. Note that $AT_{s_1}^B = AT_{s_1}$ and $RAT_{t_k}^B = RAT_{t_k}$. Ideally, $AT_{s_i}^B = RAT_{s_i}^B$ since the solution B would consume "useful slack" completely.

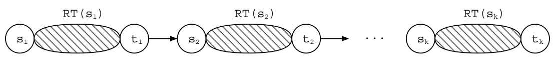


Figure 4: Example of a circuit for Theorem 1

The following theorem is to quantify the spreading of "useful slack" and it matches with our experimental results of buffering for minimum cost. We define the minimum delay to PO as $delay\_to\_PO^B s_i = RAT_{t_k}^B - RAT_{s_i}^B$ with buffering solution B and $delay\_to\_POs_i = RAT_{t_k} - RAT_{s_i}$ with optimal buffering solution for minimum delay.

Theorem 1 If the "useful slack" evenly distributes to every $RT(s_i)$ for $i = 1, \ldots, k$ in a manner that $\frac{delay\_to\_PO^B s_i}{delay\_to\_POs_i}$ is a constant among all s_i, then

$$RAT_{s_i}^B = RAT_{s_i} - \frac{(RAT_{t_k} - RAT_{s_i})(RAT_{t_k} - AT_{t_k})}{AT_{t_k} - AT_{s_1}}. \qquad (1)$$

Proof From the assumption that $\frac{delay\_to\_PO^B s_i}{delay\_to\_POs_i}$ is a constant, we have

$$\frac{RAT_{t_k}^B - RAT_{s_i}^B}{RAT_{t_k} - RAT_{s_i}} = \frac{RAT_{t_k}^B - RAT_{s_1}^B}{RAT_{t_k} - RAT_{s_1}} \qquad (2)$$

$$\frac{RAT_{s_i} - RAT_{s_i}^B}{RAT_{t_k} - RAT_{s_i}} = \frac{RAT_{s_1} - RAT_{s_1}^B}{RAT_{t_k} - RAT_{s_1}} \qquad (3)$$

$$RAT_{s_i} - RAT_{s_i}^B = \frac{(RAT_{t_k} - RAT_{s_i})(RAT_{s_1} - AT_{s_1})}{RAT_{t_k} - RAT_{s_1}} \qquad (4)$$

$$RAT_{s_i} - RAT_{s_i}^B = \frac{(RAT_{t_k} - RAT_{s_i})(RAT_{t_k} - AT_{t_k})}{RAT_{t_k} - RAT_{s_1}} \qquad (5)$$

$$RAT_{s_i} - RAT_{s_i}^B = \frac{(RAT_{t_k} - RAT_{s_i})(RAT_{t_k} - AT_{t_k})}{AT_{t_k} - AT_{s_1}} \qquad (6)$$

Equations (2)-(6) shows the derivation of Theorem 1. From (2) to (3), we substitute $RAT_{t_k}^B$ with RAT_{t_k} and subtract 1 from both side. (4) is based on the fact that $RAT_{s_1}^B = AT_{s_1}$ with zero slack and (5) is due to equal slack along the 2-pin routing trees. Finally, we have (6) because $RAT_{t_k} - RAT_{s_1} = AT_{t_k} - AT_{s_1}$ which is delay from s_1 to t_k. ■

Theorem 1 provides a method for adjusting the required arrival time of i in Figure 3 and the equation is shown in (7), where RAT_z is the required arrival time at z and AT_z is the arrival time at z based on buffer aware STA. Although the slack $RAT_z - AT_z$ is due to the path $\{a \rightsquigarrow g \to h \rightsquigarrow p \to r \rightsquigarrow z\}$ which is less than the slack at i, we can use $RAT_z - AT_z$ as a lower bound estimation. In such case, spreading the value of $RAT_z - AT_z$ over path $a \rightsquigarrow g \to i \rightsquigarrow q \to r \rightsquigarrow z$ gives a good adjustment to RAT_i and make the sink i a little bit more critical in the routing tree $RT(g)$.

$$\text{adjusted } RAT_i = RAT_i - \frac{(RAT_z - RAT_i)(RAT_z - AT_z)}{AT_z - AT_a} \qquad (7)$$

5. PATH BASED BUFFER INSERTION AND SIMULTANEOUS GATE SIZING

The framework of our PBBI algorithm provides us the flexibility in integrating gate sizing into PBBI. It is due to the fact that when we cascade several routing trees into one merged tree, input pin and output pin of a module along the processing path is treated as one single vertex v , which is a special candidate buffer location and a buffer must be inserted at v while the resistance/delay/capacitance characteristic of the buffer is derived from that of the module's input-to-output path. In this point of view, if we also consider gate sizing with n choices of size, then we can derive n special buffers according to the sizes. Thus, by restricting the VGDP algorithm to insert one and only one buffer at v choosing from the n special buffers, path based simultaneous buffer insertion and gate sizing is accomplished.

Indeed, treating the gate sizing as selecting a buffer from a buffer library may cause error because a gate can have more than one input and more than one output. When the algorithm changes the size of a gate along a path, for those input pins and output pins which are not along the path, their capacitance or resistance values change simultaneously without considering the impact resulted from the change. However, such sacrifice helps maintaining the solution quality of the processing path (which must be more critical than the unprocessed paths) and we have verified this claim by our experimental results such that the algorithm can substantially reduce overall area of buffers and gates.

5.1 Gate Sizing at the Sinks

Along a path, using path based simultaneous buffer insertion and gate sizing on the merged routing tree cannot solve the problem when the root of the merged routing tree also need gate sizing. It is especially true when the root of the merged routing tree is an output pin of a module. For example, in Figure 2(c), after we already finished buffer insertion for the merged routing tree of $RT(b)$ and $RT(i)$, if we are processing the path $\{o,s\}$, the merged routing tree is just $RT(o)$ itself. In addition to applying VGDP to $RT(o)$, we have to perform gate sizing for the module $m2$ with pins $\{j,k,o\}$.

Sizing up a gate reduces the output resistance and so the delay is reduced. At the same time, the input capacitance increases which in turn raises the delay of upstream interconnect. Such delay increase can be handled by adding a delay penalty when doing gate sizing as proposed in [1]. However, the main problem about this issue is that the increase in load capacitance may alter the other path delays of the previously buffered routing tree. In the above example, if we size up the module $m2$, the increase in input capacitance at j may increase the downstream load capacitance of $RT(i)$ and it may in turn increase the delay along the path $\{i,r\}$.

In order to fix this problem, we perform gate sizing at all sinks of the merged routing tree while we apply VGDP. If the sink vertex is not a PO, and if the module at the sink is not sized previously, we perform gate sizing for the module at the sink. At a sink s without gate sizing, VGDP starts to propagate one solution with the corresponding load capacitance, the required arrival time RAT_s, and zero cost (which means no buffer has been inserted up to s). With gate sizing at s, we propagate n solutions (n is the total number of different gate sizes), each of which have a scaled load capacitance, the gate size as its cost, and an modified required arrival time $RAT'_s(x_i)$. Assume that the original output resistance of the gate is R_0 and that of a sized gate is R_i, while R_b and C_b is the output resistance and input capacitance of the buffer used in buffer aware STA. For simplicity, we assume that there is a linear relationship between the delay change $(RAT'_s(x_i) - RAT_s)$ and the change in resistance $(R_i - R_0)$. Empirically, we found that (8) gives a good and effective calculation for $RAT'_s(x_i)$, where L_{opt} is the optimal buffer interval which is also used in [6]. Intuitively, $(R_i - R_0)(L_{opt}C + C_b)$ gives the change in delay if there exists a buffer in the downstream of s while the length between s and the buffer is L_{opt}.

$$\begin{aligned} RAT'_s &= RAT_s + (R_i - R_0)(L_{opt}C + C_b) \\ RAT'_s &= RAT_s + (R_i - R_0)\left(\sqrt{\frac{2R_bC_bC}{R}} + C_b\right) \qquad (8) \end{aligned}$$

6. EXPERIMENTAL RESULTS

circuit	Circuit Size # mod	# edge	total	# B can	min D (ns)	buf cost
a1	53	68	121	1449	10.5	204
a2	154	160	314	1880	20	274
a3	259	272	531	3190	21	484
a4	328	352	680	3897	26.5	592
a5	465	480	945	4316	43	645
a6	564	592	1156	6625	28.3	1009
a7	742	768	1510	6810	61.5	1033
a8	766	816	1582	8658	33	1314
a9	893	928	1821	8083	84	1238
a10	999	1072	2071	11623	54	1700
a11	1958	2136	4094	20432	105	2991
a12	2983	3120	6103	25738	122	3925

Table 1: Summary of testcases

We have implemented a very efficient VGDP buffer insertion algorithm according to [16] which uses approximation techniques to improve the efficiency. We have performed experiments on 12 combinational circuits which are randomly generated based on real nets from IBM. For simplicity, buffer cost refers to the number of buffer inserted, which approximately stands for buffer area. All experiments are running on a Debian Linux machine with 2.4 GHz processor and 1GB RAM. The size of each testcase is summarized in Table 1. The first column refers to the number of modulus in the circuit. The second one refers to the number of edges, which equals the total number of sinks for all routing trees. The column "total" is "mod"+"edge" which is a measure of the circuit size. "# B can" is the total number of buffer candidate locations for each circuit. For each net in the circuit, a minimum-delay buffer insertion algorithm is applied, which gives the minimum achievable delay of the circuit. The minimum delay in *ns* and the number of buffer inserted are listed in the last two columns of Table 1.

6.1 Path Based Buffer Insertion

We first compare our PBBI algorithm with the net based buffer insertion algorithm using the same implementation of VGDP. We set the delay constraint of a circuit to be its minimum achievable delay as reported in Table 1. For all the following comparisons between VGDP and PBBI, the worst slacks of the circuit for both methods are similar and are not shown in the tables. For net based buffering, we tried several different ordering[1] and made the conclusion that, on average, all tested ordering performs similarly in terms of total buffer cost. As a result, in our experimental results, we used the ordering based on ascending order of worse slack for comparison. The results are shown in Table 2. The column "B cost" stands for the total number of buffer inserted while "% redu" is the percentage of buffer cost reduction when PBBI is compared to net based buffer insertion.

	Net based		Path based (PBBI)		
circuit	B cost	CPU/s	B cost	% redu	CPU/s
a1	127	0.7	96	24.41	0.7
a2	144	0.5	138	4.17	1.1
a3	216	1.0	177	18.06	2.0
a4	317	1.3	288	9.15	2.8
a5	357	1.0	338	5.32	3.6
a6	285	2.1	232	18.60	4.3
a7	384	1.5	307	20.05	8.4
a8	434	2.6	359	17.28	5.5
a9	461	1.8	363	21.26	16.1
a10	492	3.7	414	15.85	13.6
a11	841	5.9	766	8.92	39.0
a12	1048	5.1	852	18.70	38.1
Total	5106	27.1	4330	15.20	135.2

Table 2: Comparison of net based and path based buffer insertion

From Table 2, we have the following conclusions.

- The average reduction in buffer area is 15% for PBBI algorithm when comparing to the net based buffer insertion.
- Although there is more than 5× increase in CPU-time, the total CPU time for the biggest circuit with $3k$ modules is still less than 1 minute. In fact, the CPU time is empirically linear to the size of the circuit and the buffer candidate locations.

	Net based		Path based (PBBI)		
circuit	B cost	CPU/s	B cost	% redu	CPU/s
a1	103	0.5	83	24.10	0.7
a2	128	0.5	114	12.28	1.1
a3	195	0.9	151	29.14	1.9
a4	294	1.1	241	21.99	2.8
a5	309	0.9	304	1.64	3.8
a6	272	1.8	186	46.24	4.3
a7	343	1.3	248	38.31	8.5
a8	401	2.2	314	27.71	5.7
a9	423	1.7	294	43.88	16.1
a10	463	3.3	336	37.80	13.4
a11	787	5.6	657	19.79	31.2
a12	961	5.2	712	34.97	33.9
Total	4679	25.2	3640	28.54	123.4

Table 3: Comparison of net based and path based buffer insertion considering buffer blockages

In our next experiment, we consider buffer blockages and the results are shown in Table 3. From the results, we obtained an even greater buffer cost reduction of 29% when comparing to the net based buffering. From our observation, it is due to the fact that buffer insertion with blockages becomes more complicated, and an algorithm with a global view would perform better.

[1]The ordering includes (1) the topological order from PO to PI and its reversal (2) first process the net with the smallest worse slack; (3) first process the net such that most critical paths are passing through it; (4) according to the descending order of the total load capacitance of the nets.

	Net based				Path based (PBBI+GS)				
circuit	B cost	Δ G	Δ area	CPU/s	B cost	Δ G	Δ area	% redu	CPU/s
a1	115	36	151	0.6	81	8	89	41.06	1.5
a2	189	20	209	0.5	102	24	126	39.71	2.5
a3	278	44	322	0.9	151	10	161	50.00	7.0
a4	391	72	463	1.2	222	94	316	31.75	10.3
a5	488	112	600	0.9	218	61	279	53.50	15.7
a6	641	159	800	1.9	197	16	213	73.38	18.1
a7	779	198	977	1.3	235	34	269	72.47	44.9
a8	839	166	1005	2.4	310	52	362	63.98	19.2
a9	973	310	1283	1.7	279	42	321	74.98	61.9
a10	1089	216	1305	3.5	322	39	361	72.34	38.7
a11	2097	331	2428	6.2	587	68	655	73.02	93.2
a12	3149	837	3986	6.0	709	110	819	79.45	119.4
Total	11028	2501	13529	27.2	3413	558	3971	70.65	432.5

Table 4: Comparison of net based and path based buffer insertion considering gate sizing

6.2 Simultaneous Gate Sizing and Buffer Insertion

We have performed the similar experiments for simultaneously gate sizing and buffer insertion. For net based approach, we have implemented the delay penalty scheme [1] which includes driver sizing into net based buffer insertion process. However, the delay penalty formula used in the paper is mainly for solution comparison in VGDP. When applying the delay penalty as a mean to estimate the delay increase in upstream interconnect, we have to scale the delay penalty empirically. The results is shown in Table 4. In the table, "Δ G" is the total size change in all gates, "Δ area" is the total increase in cost (area) from the buffer insertion/gate sizing, and "% redu" is the percentage of total cost reduction when PBBI+GS is compared to the net based approach.

From Table 4, we found that the overall cost reduction by PBBI algorithm is 71% when comparing to net based gate sizing/buffer insertion. As the same time, our PBBI+GS algorithm is reasonably efficient as the runtime is within 2 minutes for the biggest testcase.

Our novel technique of performing gate sizing at sink results in significant improvement in solution quality. For [1], the delay penalty is calculated at the driver of a routing tree. It is simple and effective but it only reflects the delay increase in the upstream interconnect towards the gate and the routing tree itself. We have performed experiments for our PBBI algorithm using different gate sizing schemes, results show that gate sizing using delay penalty takes 14% more area than our gate sizing at sink technique (the table is not shown due to space limit). In conclusion, the lack of a global view results in the trapping into a local optimal solution and the error exacerbates when the complexity of the problem increases.

7. CONCLUSION

The VLSI technology scaling requests increasingly more buffers in circuit designs and therefore buffer insertion needs to be carried in more elaborated manner. As a departure from traditional net based buffer insertion methods, we propose a path based buffer insertion approach which can obtain a better buffer usage efficiency due to its global view. To the best of our knowledge, this is the first work on path based buffer insertion. Compared to network based methods, our approach is more practical in terms of computation complexity. Several techniques are proposed along with the path based buffering including buffer aware static timing analysis, slack spreading along off-paths and simultaneous sink sizing. Experimental results show that our approach can reduce buffer/gate cost by 71% on average compared to net based methods.

8. REFERENCES

[1] C. J. Alpert, C. Chu, G. Gandham, M. Hrkic, J. Hu, C. Kashyap, and S. T. Quay. Simultaneous driver sizing and buffer insertion using delay penalty estimation technique. *IEEE Trans. on CAD*, 23(1):136–141, January 2004.

[2] C. J. Alpert and A. Devgan. Wire segmenting for improved buffer insertion. In *Proc. of DAC*, pages 588–593, 1997.

[3] C. J. Alpert, A. Devgan, and S. T. Quay. Buffer insertion for noise and delay optimization. In *Proc. of DAC*, pages 362–367, 1998.

[4] C. J. Alpert, A. Devgan, and S. T. Quay. Buffer insertion with accurate gate and interconnect delay computation. In *Proc. of DAC*, pages 479–484, 1999.

[5] C. J. Alpert, M. Hrkic, J. Hu, and S. T. Quay. Fast and flexible buffer trees that navigate the physical layout environment. In *Proc. of DAC*, pages 24–29, 2004.

[6] C. J. Alpert, J. Hu, S. S. Sapatnekar, and C. N. Sze. Accurate Estimation of Global Buffer Delay within a Floorplan. In *Proc. of ICCAD*, pages 706–711, 2004.

[7] H. B. Bakoglu. *Circuits, interconnections and packaging for VLSI.* Addison-Wesley, Reading, MA, 1990.

[8] C. C. N. Chu and D. F. Wong. A quadratic programming approach to simultaneous buffer insertion/sizing and wire sizing. *IEEE Trans. on CAD*, 18(6):787–798, June 1999.

[9] C. C. N. Chu and D. F. Wong. Closed form solution to simultaneous buffer insertion/sizing and wire sizing. *ACM Trans. on Design Automation of Electronic Systems*, 6(3):343–371, July 2001.

[10] J. A. Davis, R. Venkatesan, A. Kaloyeros, M. Beylansky, S. J. Souri, K. Banerjee, K. C. Saraswat, A. Rahman, R. Reif, and J. D. Meindl. Interconnect limits on gigascale integration (GSI) in the 21st century. *Proc. of IEEE*, 89(3):305–324, March 2001.

[11] S. Dhar and M. A. Franklin. Optimum buffer circuits for driving long uniform lines. *IEEE Journal of Solid-State Circuits*, 26(1):32–38, January 1991.

[12] L. P. P P. van Ginneken. Buffer placement in distributed RC-tree networks for minimal Elmore delay. In *Proc. of ISCAS*, pages 865–868, 1990.

[13] C. V. Kashyap, C. J. Alpert, F. Liu, and A. Devgan. Closed-form expressions for extending step delay and slew metrics to ramp inputs for RC trees *IEEE Trans. on CAD*, 23(4):509–516, April 2004.

[14] Y. Jiang, S. S. Sapatnekar, C. Bamji, and J. Kim. Interleaving buffer insertion and transistor sizing into a single optimization. *IEEE Trans. on VLSI Systems*, 6(4):625–633, December 1998.

[15] Y.-C. Ju, and R. A. Saleh. Incremental techniques for the identification of statically sensitizable critical paths. In *Proc. of DAC*, pages 541–546, 1991.

[16] Z. Li, C. N. Sze, C. J. Alpert, J. Hu, and W. Shi. Making fast buffer insertion even faster via approximation techniques. In *Proc. of ASPDAC*, pages 13-18, 2005.

[17] J. Lillis, Algorithms for Performance Driven Design of Integrated Circuits. *PhD Thesis, UC San Diego*, August 1996.

[18] J. Lillis, C. K. Cheng, and T. Y. Lin. Optimal wire sizing and buffer insertion for low power and a generalized delay model. *IEEE Journal of Solid-State Circuits*, 31(3):437–447, March 1996.

[19] I.-M. Liu, A. Aziz, and D. F. Wong. Meeting delay constraints in DSM by minimal repeater insertion. In *Proc. of DATE*, pages 436–441, 2000.

[20] I.-M. Liu, A. Aziz, D. F. Wong, and H. Zhou. An efficient buffer insertion algorithm for large networks based on Lagrangian relaxation. In *Proc. of ICCD*, pages 614–621, 1999.

[21] S. S. Sapatnekar. Timing. *Kluwer Academic Publishers*, 2004.

[22] P. Saxena, N. Menezes, P. Cocchini, and D. A. Kirkpatrick. Repeater scaling and its impact on CAD. *IEEE Trans. on CAD*, 23(4):451–463, April 2004.

[23] W. Shi and Z. Li. An $O(n \log n)$ time algorithm for optimal buffer insertion. In *Proc. of DAC*, pages 580–585, 2003.