Fast Interconnect Synthesis with Layer Assignment

Zhuo Li
IBM Research
Austin, Texas 78759
lizhuo@us.ibm.com

Charles J. Alpert
IBM Research
Austin, Texas 78759
alpert@us.ibm.com

Shiyan Hu
Michigan Technological University
Houghton, MI 49931
shiyan@mtu.edu

Tuhin Muhmud
IBM Systems and Technology Group
Austin, Texas 78759
tuhinm@us.ibm.com

Stephen T. Quay
IBM Systems and Technology Group
Austin, Texas 78759
quayst@us.ibm.com

Paul G. Villarrubia
IBM Systems and Technology Group
Austin, Texas 78759
pgvillar@us.ibm.com

ABSTRACT

As technology scaling advances beyond 65 nanometer node, more devices can fit onto a chip, which implies continued growth of design size. The increased wire delay dominance due to finer wire widths makes design closure an increasingly challenging problem. Interconnect synthesis techniques, such as buffer insertion/sizing and wire sizing, have proven to be the critical part of a successful timing closure optimization tool.

Layer assignment, which was traditionally treated as same as wire sizing, is more effective and friendly in the design flow than wire sizing in the advanced technologies. Techniques for simultaneous layer assignment and buffer insertion with resource control are increasingly important for the quality of results of interconnect synthesis. This paper outlines the importance of layer assignment over wire sizing, and presents efficient techniques to perform concurrent buffer insertion and layer assignment to fix the electrical and timing problems, while maintaining speed and efficient use of resources.

Categories and Subject Descriptors

B.7.2 [**Integrated Circuits**]: Design Aids—*Placement and routing*

General Terms

Algorithms

Keywords

Buffer Insertion, Interconnect Synthesis, Layer Assignment, Wire Sizing

1. INTRODUCTION

With the continued scaling of process technologies, the transistor feature size and signal wire dimension continues to decrease. Even with use of copper wires and low dielectric constant materials, and improved metal aspect ratio, the interconnect performance is still more dominant than logic performance [1]. A design that meets timing in a zero wireload timing environment (e.g., after logic synthesis) will no longer meet timing after performing physical synthesis. Interconnect synthesis, a series of timing optimiation techniques including buffer insertiong/sizing, wire sizing, and layer assignment, have been widely used in design automation tools to achieve design closure.

Buffer (repeater) insertion, is a popular technique to reduce interconnect delay and fix electrical violations. It effectively divides the wire into small segments that makes the interconnect delay almost linear in terms of wire length, and decouples the capacitive loads of non-critical sinks from the critical paths. The number of block-level nets requiring buffer insertion rises as process technology scales down and a larger number of buffers are required to meet interconnect timing requirements (upwards of 25-50% of the logic may consist of repeaters) [2]. Perhaps the most well-known buffer insertion algorithm is Van Ginneken's classic dynamic programming algorithm [3], which has led to several extensions [4, 5].

Wire sizing reduces interconnect resistance by increasing wire widths, while also brings the additional capacitance. Depending on driven strengths, interconnect topology and parasitics, careful wire sizing could reduce the interconnect delay. The wire sizing literature was very active in the 1990s (e.g., [6, 7]) and it has been combined with buffer insertion [4, 1, 8] to be more effective.

Though buffer insertion and wire sizing have been extensively studied, the current interconnect synthesis methodology is not complete due to the mis-treatment of layer assignment. While layer assignment and wire sizing are generally studied together, they have fundamental difference.

1.1 Wire Sizing

Previous interconnect synthesis with wire sizing can be classified intro three categories as shown in Fig. 1, namely (a) simultaneous buffer insertion and continuous wire sizing, (b) simultaneous buffer insertion and discrete tapered wire sizing (TWS), and (c) simultaneous buffer insertion and uniform wire sizing (UWS).

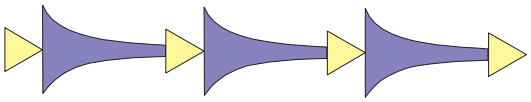
Continous wire sizing allowed the wire width at any location along a wire to be a different number, therefore achieved the theoretical lower bound on delay of a single wire connecting 2 pins. Based on Elmore delay model, Fishburn and Schevon [7] and Chen *et al.* both discovered the optimal wire shape function is an exponential curve. The optimal shape of a wire was fat near the source and gradually narrowd towards the sink the net. When compined

ISPD'08, April 13–16, 2008, Portland, Oregon, USA.

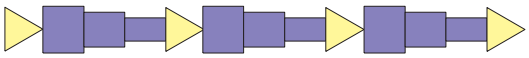
with buffer insertion, Alpert *et al.* showed that equal distance between buffers will enforce optimal solution.

Since continous wire shape is not manufacturable, several results for discrete tapered wire sizing (TWS), which divides the wire into small segments and assign the entire segment with same wire size, were developed. A good example is Cong *et al.* [9]'s demonstration of the dominance property, which was effectively extended to optimize different delay and area objectives and different delay models [6, 10]. Several works have also studied the simultaneous optimization of buffer insertion/sizing and wire sizing, including closed-form solutions for a single wire [8], algorithms for tree topologies [4], and delay model considering fringing capacitance [11].

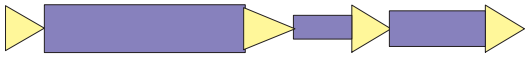
Later Alpert *et al.* [12] argued that tapering caused headaches for the router and that one could achieve similar quality results by simply increasing the width of the entire wire, as long as the result was combined with buffering. With uniform wire sizing (UWS), the authors of [12] presented a new polynomial optimal algorithm for simultaenous buffer insertion and wire sizing, which is better than pseudopolynomial algorithm proposed in [4], and extended the algorithm to resource balancing, which is known to be computationally expensive even with buffer insertion alone [13].



(a) Simultaneous buffer insertion and continous wire sizing.



(b) Simultaneous buffer insertion and taperd wire sizing (TWS).



(c) Simultaneous buffer insertion and uniform wire sizing (UWS).

Figure 1: Three types of interconnect synthesis.

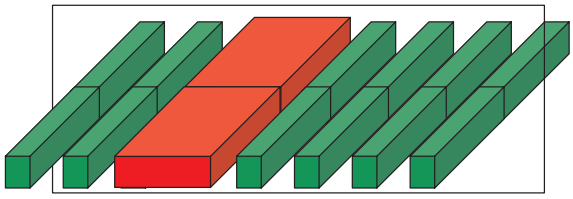


Figure 2: The example of wire sizing using multiple routing tracks.

This entire set of literature starts with the same set of assumptions, that the resistance was inversely proportional to the wire area, that capacitance was proporitonal to the wire area (plus fringing capacitance). In other words, the wire's layer was predetermined. Under this scenario, even with the most limiting assumptions of Alpert *et al.*, wire sizing still had a cost, namely that the more wires sized the fewer routing tracks were available for routing. For example, sizing a wire up to three routing tracks from one routing track may reduce wire delay significantly, but it also removes two potentially precious routing tracks, as shown in Fig. 2. Thus, during physical synthesis [14], one has to either be extremely conservative so as not to hurt routability or iterate with a router, which could be prohibitively expensive.

1.2 Layer Assignment

Starting with the 90 nm node, "wire sizing" problem changed with the availability of a top layer of "thick metal". The wires at the top layer have significantly less resistance than the other available metals, while have roughly constant capacitance. With 65 nm node, there is a thick metal and a "super thick" metal available as shown in Fig. 3. With the 45 nm node, there are four different layers. [1]

Fig. 3 shows a cross section of a routine tile with 14 routing tracks in 65 nm node, of which eight belong to a single 1x thickness layer, four belong to a double 2x thickness layer, and two belong to top 4x thickness layer. If one wants to assign a wire in this tile to the thickest metal to get the best performance, there are two tracks of thickest metal available to choose from, and this assignment does not consume an extra routing resources!

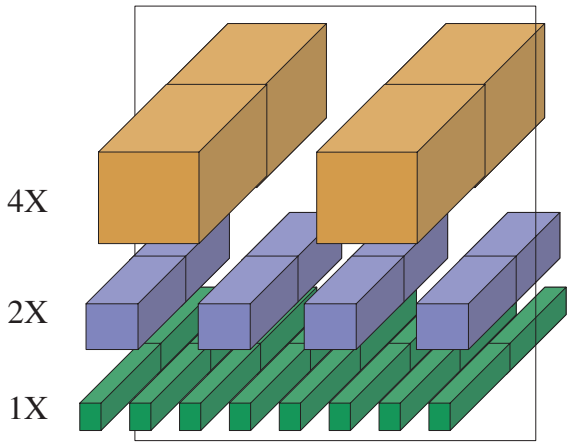


Figure 3: Routing track for cu65nm and beyond.

Layer assignment reduces both interconnect delay and the number of buffers needed. As shown in Fig. 3, the wire on thicker layer is both wider and thicker. Assuming that wire width changes at the same ratio as the wire thickness, resistance per unit length on 2x thickness layer has roughly 1/4 of resistance per unit length on 1x thickness layer, and resistance per unit length on 4x thickness layer has roughly 1/16 of the one on the 1x thickness layer. On another hand, the distance between wires also increases as the wire is assigned to thicker layer, the self and fringing capacitance per unit length are roughly constant. From [12], the expected distance between buffers L is

$$L = K_1 r_g c_g / \sqrt{r_w c_w},$$

and the optimal buffer delay per unit length T is

$$T = K_2 \sqrt{r_g c_g r_w c_w},$$

where K_1 and K_2 are constants, r_g and c_g are unit area buffer driving resistance and input capacitance, r_w is the sheet resistance per

[1] In this paper, a "layer" is defined based on its parasitics since what really matters for layer assignment is the ability to switch parasitics. For example, if there are two horizontal layer and one vertical layer with similar parasitics, we still refer them as one layer. There needs to be a minimum of two layers, which makes 90 nm the first meaningful technology node for layer assignment.

unit length and c_w is the wire capacitance per unit length. Assuming constant wire capacitance, the buffer spacing can roughly go 2x further for 2x thickness layer, and 4x further for 4x thickness layer. Signal travels 2x faster for 2x thickness layer and 4x faster for 4x thickness layer. In reality, with the impact of input slew, non-zero buffer intrinsic delay, wire capacitance change, the above number could be off but the benefit of layer assignment is still tremendous. For example, for 4x thickness layer, our test-cases show that the buffer spacing can go 3.5x further, and signal can travel 3x faster.

The problem now becomes to perform a global layer assignment of wires to each level of metal, and the delays incurred by this assignment need to be taken into account during physical synthesis, which means they need to happen before routing. Naturally, this step needs to be coupled with buffer insertion as shown in the above discussions, where wire resources, buffer area/power, and timing need to be balanced and may be tuned for each particular design.

2. INTEGRATED BUFFER INSERTION AND LAYER ASSIGNMENT

In this paper, we propose two techniques for simultaneous buffer insertion and layer assignment.

- **LADY**. Layer Assignment for Delay Improvement. This technique is mainly used in critical path optimization and histogram compression stages [14].
- **LASR**. Layer Assignment for Slew Recovery. This technique extends the slew buffering algorithm [15] to fix the capacitance and slew violations during electrical correction stage [14].

Similar to uniform wire sizing [12], layer assignment in this paper will assign the entire wire to the same layer. Both algorithms are embedded into dynamic programming style Van Ginneken's buffer insertion framework, and are designed to minimize buffer cost and usage of the thicker metal. We want to emphasize that it is very important to control both buffer cost and wire resources on thicker layers.

- Buffer resources control. Minimizing buffer cost while meeting timing or electrical constraints can help to reduce the silicon area, routing area and chip powers. With the cost consideration, van Ginnken's algoirthm extended by Lillis *et al.* [4] and Hu *et al.* [15] becomes practical and necessary yet more runtime intensive [13, 15].
- Wire resources control. Minimizing the usage of the thicker metal is also necessary. Layer assignment is more like a "directive" to the router. In general, the router has the freedom to put a wire on whatever metal it wishes. So a net on 1x thickness layer could end up on 2x thickness layer. In general, that will only improve timing. But when performing layer assignment we are requiring the router to put the net on 2x or 4x thickness layer, which restricts router flexibility. So we need to be somewhat careful not to choke the router, which is why we only want to put wires up there that truly needs it and allow the router to choose for the rest of them.

Our new techniques efficiently improve the design quality with better buffer and wire resource control compared to previous method. They have been successfully embedded in IBM's physical synthesis tool called PDS [14], and show almost no runtime impact on large multi-million gate designs. The detail of problem formulations and algorithms are described in the following sections.

3. LAYER ASSIGNMENT FOR DELAY IMPROVEMENT (LADY)

3.1 Problem Formulation

Here is the simultaneous buffer insertion and layer assignment for delay improvment problem:

Buffer Insertion Considering Layer Assignment for Delay Improvement: Given a binary routing tree $T = (V, E)$, candidate buffer locations, a buffer library B, a set of routing layers, to compute a buffering and layer assignment solution such that the timing constraint is satisfied and the total buffer and wire cost is minimized.

Perhaps the naive way of performing simultaneous buffer insertion and layer assignment for delay improvement is to assign the whole net to each layer, run van Ginneken's algoirthm, and pick the best one from all solutions. The problem is that for 2-pin nets with blockages or multi-pin nets , where buffers are non-uniformly inserted, both short wires and long wires are assigned to the same layer. While assigning the long wire to lower layer could kill the performance, assigning the short wire to thicker layer will be suboptimal due to Via resistance. Also, if too many short wires are assigned to thicker layer, it will create routing congestion problem.

One could also apply the works in [4] and [12] directly to layer assignment. However, both cost function to trade off layers are not intuitive and have limitations. In [4], wiring area is treated as wire cost, and is added up directly to the buffer cost. However, utilizaton of area on thicker layers could be more costly than lower layers and finding the appropriate tradeoff between buffer cost and wire area is difficult.

In [12], three steps are taken to control wire cost, 1) each layer is preassigned a cost; 2) the layer cost of a solution is measured as the maximum cost of all used layer cost of this solution; 3) the total cost of a solution is a linear combination of layer cost and buffer cost, while the coefficient is a non-linear function of a balance parameter. The preassigned layer cost and balance parameter are not intuitive to tune by designers. The "max" cost function will over-consume wire resources of thicker layer since as long as one subnet in this solution is assigned to a particular thicker layer, all other subnets are "free" to be assigned to this layer.

Since both algorithms will directly update the cost of the solutions and create extra cost values, there could also be significant runtime overhead.

3.2 Main Idea

The algorithm of LADY extends the classic van Ginneken/Lillis [3, 4] algorithm for timing-driven buffering. Compared to other approaches, LADY avoids using wire cost which is difficult to model in practice to control wire resources. Instead, the wire resources is directly controled by timing improvement.

The main idea is that during buffering, the subnet on current layer is bumped up to the next thicker layer, if such assignment significantly improves the slack by a user-defined delay improvement threshold of current layer. Therefore, assigning a wire to thicker layer is directly controled by the timing improvement before/after layer assignment and this threshold of the current layer. For different layers, there could be different thresholds. For example, the threshold of 2x thickness layer that controls the promotion from 2x to 4x thickness layer, could be larger than the threshold of 1x thickness layer, which controls the promotion from 1x to 2x thickness layer. Wire resources are directly mapped to the timing (e.g., assign to thicker layer only if it improves the slack by 10 ps). This idea generally guarantees only long wires get promoted to thicker layer since Via may make delay worse for short wires.

An example of the algorithm is shown in Fig. 4. With 3 layers available, starting from the sink, we have 3 temporary partial solutions when the first buffer is added at node v_1. The delay of the wire are 18, 10 and 7 ps for 1x layer, 2x layer and 4x layer respectively. If the delay improvement thresholds X for both 1x and 2x layers are 5 ps, then the wire is assigned to 2x layer since the delay improvement from 1x to 2x layer is 8 ps, but the one from 2x to 4x is only 3 ps. For another set of partial solutions at node v_2, the wire is assigned to 4x layer since the delay improvement from 2x layer to 4x layer is 8 ps. It is clear that the wire needs to travel longer to be assigned to thicker layer.

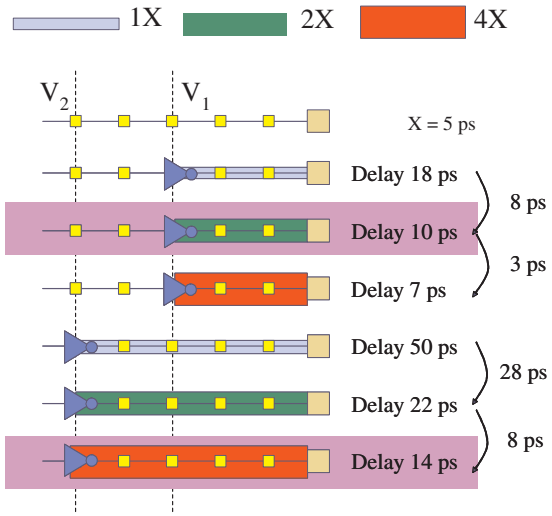


Figure 4: The example of LADY algorithm. The delay improvement threshold X is 5 ps.

3.3 The Algorithm

For completeness, some details of van Ginneken/Lillis approach are also included.

Suppose that a set of t routing layers $l_1, l_2, \ldots, l_t$ are given where l_1 is the thinner layer and l_t is the thickest layer. Each layer l_i is associated with a user-defined delay improvement threshold X_i. The LADY algorithm proceeds in a bottom-up fashion from sinks toward the driver along a given routing tree. A set of candidate solutions is propagated during the process. At driver, the solution satisfying the timing constraint and with the smallest cost is selected. Each solution is associated with a three-tuple $(\mathbb{C}, W, \mathbb{Q})$, where $\mathbb{C}$ is an array which denotes the set of the downstream capacitance for each layer, W denotes the cost of the solution and $\mathbb{Q}$ is an array which denotes the required arrival time (RAT) of each layer. In another word, $\mathbb{C}[i]$ denotes the downstream capacitance and $\mathbb{Q}[i]$ denotes the RAT corresponding to layer i for $1 \leq i \leq t$, respectively. Initially, there is a solution at each sink where $W = 0$, $\mathbb{Q}[i]$ is set to the required arrival time of the sink and $\mathbb{C}[i]$ is set to the sink capacitance for $1 \leq i \leq t$. Three operations are performed during solution propagation.

"Add Wire": To propagate a solution γ_v at position v to an upstream position u with no branching point in between, the new solution γ_u can be computed as

$$\begin{array}{lcl} \mathbb{C}(\gamma_u)[i] & = & \mathbb{C}(\gamma_v)[i] + C_{e[i]}, \\ W(\gamma_u) & = & W(\gamma_v), \\ \mathbb{Q}(\gamma_u)[i] & = & \mathbb{Q}(\gamma_v)[i] - D_{e[i]}, \end{array} \quad (1)$$

where $e = (u, v)$ and $C_{e[i]}$ denotes the capacitance of edge e in layer i and $D_{e[i]}$ denotes the Elmore delay of edge e in layer i for $1 \leq i \leq t$. Clearly, in a solution, C, Q will be updated only for wires at the same layer. Thus, no wire tapering is allowed which it is desired as shown in [12].

"Add Buffer": To add a buffer b at u to γ_u, $\gamma_{u,buf}$ can be computed as

$$\begin{array}{lcl} \mathbb{C}(\gamma_{u,buf})[i] & = & C_b, \\ W(\gamma_{u,buf}) & = & W(\gamma_u) + W_b, \\ \mathbb{Q}(\gamma_{u,buf})[i] & = & \mathbb{Q}(\gamma_u)[i] - D_b \end{array} \quad (2)$$

where D_b is the buffer delay of b and $1 \leq i \leq t$. To explore the power of layer assignment, immediate downstream wires will be promoted to the next thicker layer, which is the critical difference between standard van Ginneken/Lillis algorithm and our new LADY algorithm. To accomplish this, for the current layer i, the effect of layer promotion is characterized by $C[i+1], Q[i+1]$ for $1 \leq i < t$. Such a layer promotion is accepted if it buys us slack improvement X_i. For example, for $i = 1$, if $Q[2] - Q[1] \geq X_1$, the promotion will be accepted. This process will be repeated until it cannot lead to significant timing improvement or the wire is at the thickest layer. The layer will be recorded after a wire is promoted.

Algorithm: Layer Assignment for Delay.
Input: T: routing tree, B: buffer library
Output: timing buffering solution with layer assignment

1. for each sink s, build a solution set $\{\gamma_s\}$, where $W(\gamma_s) = 0$, $\mathbb{Q}(\gamma_s)[i]$ is set to RAT of s, and $\mathbb{C}(\gamma_s)[i] = C_s$ for each layer i
2. for each branching point/driver v_t in the order given by a postorder traversal of T, let T' be the two branches T_1, T_2 of v_t and Γ' be the solution set corresponding to T', do
3. for each wire e in T', in a bottom-up order, do
4. for each $\gamma \in \Gamma'$ corresponding to T', do
5. for each layer i, $1 \leq i \leq t$, do
6. set $\mathbb{C}(\gamma)[i] = \mathbb{C}(\gamma)[i] + C_{e[i]}$
7. set $\mathbb{Q}(\gamma)[i] = \mathbb{Q}(\gamma)[i] - D_{e[i]}$
8. LADYUpdateSolutionSet (γ, Γ')
9. if the current position allows buffer insertion, then
10. for each buffer type $b_j \in B$, do
11. for each $\gamma \in \Gamma'$, generate a new solution γ'
12. for each layer i, do
13. set $\mathbb{C}(\gamma')[i] = C_{b_j}$
14. set $W(\gamma') = W(\gamma) + W_{b_j}$
15. set $\mathbb{Q}(\gamma')[i] = \mathbb{Q}(\gamma)[i] - D_{b_j}$
16. LADYUpdateSolutionSet (γ', Γ')
17. for each layer $i = 2$ to t,
18. for each $\gamma \in \Gamma'$,
19. if $\mathbb{Q}[i] - \mathbb{Q}[i-1] < T_d$,
20. record $i - 1$ and break the loops
21. // merge Γ_1 and Γ_2 to Γ_{v_t}
22. set $\Gamma_{v_t} = \emptyset$
23. for each $\gamma_1 \in \Gamma_1$ and $\gamma_2 \in \Gamma_2$, generate a new solution γ'
24. for each layer i, do
25. set $\mathbb{C}(\gamma')[i] = \mathbb{C}(\gamma_1)[i] + \mathbb{C}(\gamma_2)[i]$
26. set $W(\gamma') = W(\gamma_1) + W(\gamma_2)$
27. set $\mathbb{Q}(\gamma')[i] = \min\{\mathbb{Q}(\gamma_1)[i], \mathbb{Q}(\gamma_2)[i]\}$
28. LADYUpdateSolutionSet (γ', Γ_{v_t})
29. return γ satisfying timing constraint with the smallest cost at driver

Figure 5: Layer assignment for delay.

"Branch Merge": Suppose that solutions of left branch T_l and right branch T_r at a branching point v_t are obtained. Denote the left-branch solution set and the right-branch solution set by Γ_l and Γ_r, respectively. To merge them, for each solution $\gamma_l \in \Gamma_l$ and

each solution $\gamma_r \in \Gamma_r$, generate a new solution γ' as

$$\begin{array}{lcl} \mathbb{C}(\gamma')[i] & = & \mathbb{C}(\gamma_l)[i] + \mathbb{C}(\gamma_r)[i], \\ W(\gamma') & = & W(\gamma_l) + W(\gamma_r), \\ \mathbb{Q}(\gamma')[i] & = & \min\{\mathbb{Q}(\gamma_l)[i], \mathbb{Q}(\gamma_r)[i]\}. \end{array} \quad (3)$$

Note that only wires in the same layer can be merged.

To accelerate the algorithm, pruning will be performed as in van Ginneken/Lillis algorithm. For any two solutions γ_1, γ_2 *at the same node*, γ_1 will be pruned if $\mathbb{C}(\gamma_1)[1] \leq \mathbb{C}(\gamma_2)[1], W(\gamma_1) \leq W(\gamma_2)$ and $\mathbb{Q}(\gamma_1)[1] \geq \mathbb{Q}(\gamma_2)[1]$. In our implementation, solutions are compared by the characteristics of default layer. Comparing every layer is not only slow but unnecessary. The motivation is that if the downstream capacitance on default layer is smaller, then in general the capacitance on thicker layer is smaller too. With the same cost as default layer, the only difference is the slack relationship. But with the anticipation of Via resistance cost from the lower layer to the upper layer, we can still prune based on the default layer with the spirit like aggressive predictive pruning in [13, 16]. Our pruning strategy can effectively accelerate the approach while keeping the solution quality. Note that range search tree can be used for efficient pruning [4].

Please Refer to Fig. 5 and Fig. 6 for the pseudo-code of LADY.

Procedure: LADYUpdateSolutionSet
Input: γ': a candidate solution, Γ: a solution set
Output: an updated solution set Γ

1. for each solution $\gamma \in \Gamma$, do
2. if $\mathbb{C}(\gamma)[1] \leq \mathbb{C}(\gamma')[1], W(\gamma) \leq W(\gamma')$ and $\mathbb{Q}(\gamma)[1] \geq \mathbb{Q}(\gamma')[1]$,
3. return Γ
4. if $\mathbb{C}(\gamma')[1] \leq \mathbb{C}(\gamma)[1], W(\gamma') \leq W(\gamma)$ and $\mathbb{Q}(\gamma')[1] \geq \mathbb{Q}(\gamma)[1]$,
5. remove γ from Γ
6. insert γ' into Γ and return Γ

Figure 6: Procedure of updating solution set for LADY.

4. LAYER ASSIGNMENT FOR SLEW RECOVERY (LASR)

4.1 Problem Formulation

In [15], a new buffer insertion problem and its corresponding algorithm are proposed to find a solution with minimize buffer resources to meet the slew constraints. The idea is that most of nets in the design will not be timing critical or show better timing behavior after purely slew based buffering, while the resources are minimal. Also, since slew constraint is a local effect, the new algorithm is much faster than delay based buffering algorithm.

However, due to existance of placement blockages (such as IP, macros, decoupling capactiance) many slew violations can not be fixed without layer assignment. Traditionally, when layer assignment is not available, such scenario will force buffer insertion algorithm to try "slew recovery", which means even no solution satisfies the slew constraint, we need to generate the best possible solutions (e.g., putting buffers at both end of blockages) instead of giving up and returning no solutions at all. With layer assignment, the slew recovery problem can be solved more effectively. We also want to minimize wire usage on thicker metal as described at the beginning of Section 2.

Here is the simultaneous buffer insertion and layer assignment for slew recovery problem:

Buffer Insertion Considering Layer Assignment for Slew Recovery: Given a binary routing tree $T = (V, E)$, candidate buffer locations, a buffer library B, a set of routing layers, to compute a buffering and layer assignment solution such that the slew constraint is met and minimum amount of thicker layer wire resources are used.

4.2 Main Idea

The main idea of LASR algorithm is that during buffering, when a route goes over a blockage, bump the subnet up to the lowest thicker layer that is necessary to meet the slew target. In such way, LASR guarantees to use the minimum thicker layer wire resources.

An example of the algorithm is shown in Fig. 7. Starting from the sink, the algorithm finds that there is no feasible solution to meet the slew target for blockage B and D with default layer, therefore assigns the subnets to 2x layer and is able to fix the slew violations. For blockage A, wire needs to be assigned to 4x layer to meet the slew target since the blockage is too long to cross with 2x layer. It is clear that LASR uses minimum resources (e.g., wire is assigned to 1x layer to cross the small blockage C).

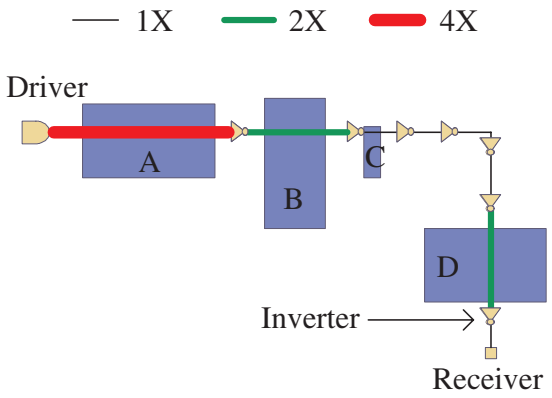


Figure 7: The example of LASR algorithm. Blue rectangles are blockages. Green wires represents 2x layer and red wires represents 4x layer.

4.3 The Algorithm

The algorithm of LASR significantly extends the slew buffering algorithm proposed in [15]. The critical difference between the new approach and [15] is the judicious usage of layer assignment when a large blockage is encountered. The algorithm in [15] may simply fail with a large blockage. In contrast, our approach will explore the power of layer assignment to recover the slew violations and compute the best possible solution. For completeness, some details of the algorithm in [15] are included.

Recall that a set of t routing layers $l_1, l_2, \ldots, l_t$ are given where l_1 is the lowest layer and l_t is the thickest layer. In the algorithm, a set of candidate solutions are propagated in the bottom-up fashion, i.e., from the sinks to the source along the given routing tree. Each candidate solution γ is characterized by a three-tuple $(\mathbb{C}, W, \mathbb{S})$, where $\mathbb{C}$ is an array which denotes the set of downstream capacitance for each layer, W denotes the cost of the solution, and $\mathbb{S}$ is an array which denotes the set of the *cumulative slew degradation along wire* for each layer. In another word, $\mathbb{C}[i]$ denotes the downstream capacitance and $\mathbb{S}[i]$ denotes the cumulative slew degradation along wire of layer i for $1 \leq i \leq t$, respectively. Note that cumulative slew degradation along wire is defined by Bakoglu's metric [17] as $S(e) = \ln 9 \cdot D(e)$ for an edge e with Elmore delay $D(e)$. Initially, there is a solution at each sink where $W = 0$,

$\mathbb{S}[i] = 0$ and $\mathbb{C}[i]$ is set to the sink capacitance for $1 \leq i \leq t$. Three operations are performed during solution propagation.

Algorithm: Layer Assignment for Slew Recovery.

Input: T: routing tree, B: buffer library, α: slew constraint
Output: slew buffering solution with layer assignment

1. for each sink s, build a solution set $\{\gamma_s\}$, where $W(\gamma_s) = 0$, $\mathbb{S}(\gamma_s)[i] = 0$, and $\mathbb{C}(\gamma_s)[i] = C_s$ for each layer i
2. for each branching point/driver v_t in the order given by a postorder traversal of T, let T' be the two branches T_1, T_2 of v_t and Γ' be the solution set corresponding to T', do
3. for each wire e in T', in a bottom-up order, do
4. for each $\gamma \in \Gamma'$ corresponding to T', do
5. for each layer i, $1 \leq i \leq t$, do
6. set $\mathbb{C}(\gamma)[i] = \mathbb{C}(\gamma)[i] + C_{e[i]}$
7. set $\mathbb{S}(\gamma)[i] = \mathbb{S}(\gamma)[i] + \ln 9 \cdot D_{e[i]}$
8. LASRUpdateSolutionSet (γ, Γ')
9. if the current position allows buffer insertion, then
10. for each buffer type $b_j \in B$, do
11. for each $\gamma \in \Gamma'$, generate a new solution γ'
12. for each layer i, do
13. set $\mathbb{C}(\gamma')[i] = C_{b_j}$
14. set $W(\gamma') = W(\gamma) + W_{b_j}$
15. set $\mathbb{S}(\gamma')[i] = 0$
16. LASRUpdateSolutionSet (γ', Γ')
17. for each layer $i = 1$ to t,
18. for each $\gamma \in \Gamma'$,
19. if $\sqrt{S(\gamma)[i]^2 + (R_b \cdot C(\gamma)[i] + K_b)^2} \leq \alpha$,
20. record i and break the loops
21. if $i = t$,
22. choose solutions with slew violations smaller than T_s
23. // merge Γ_1 and Γ_2 to Γ_{v_t}
24. set $\Gamma_{v_t} = \emptyset$
25. for each $\gamma_1 \in \Gamma_1$ and $\gamma_2 \in \Gamma_2$, generate a new solution γ'
26. for each layer i, do
27. set $\mathbb{C}(\gamma')[i] = \mathbb{C}(\gamma_1)[i] + \mathbb{C}(\gamma_2)[i]$
28. set $W(\gamma') = W(\gamma_1) + W(\gamma_2)$
29. set $\mathbb{S}(\gamma')[i] = \max\{\mathbb{S}(\gamma_1)[i], \mathbb{S}(\gamma_2)[i]\}$
30. LASRUpdateSolutionSet (γ', Γ_{v_t})
31. return feasible solution γ with the smallest cost at driver

Figure 8: Layer assignment for slew recovery.

Procedure: LASRUpdateSolutionSet

Input: γ': a candidate solution, Γ: a solution set
Output: an updated solution set Γ

1. for each solution $\gamma \in \Gamma$, do
2. if $\mathbb{C}(\gamma)[1] \leq \mathbb{C}(\gamma')[1]$, $W(\gamma) \leq W(\gamma')$ and $\mathbb{S}(\gamma)[1] \leq \mathbb{S}(\gamma')[1]$,
3. return Γ
4. if $\mathbb{C}(\gamma')[1] \leq \mathbb{C}(\gamma)[1]$, $W(\gamma') \leq W(\gamma)$ and $\mathbb{S}(\gamma')[1] \leq \mathbb{S}(\gamma)[1]$,
5. remove γ from Γ
6. insert γ' into Γ and return Γ

Figure 9: Procedure of updating solution set for LASR.

"Add Wire": To propagate a solution γ_v from a node v to its parent node u through edge $e = (u, v)$. Denote by $e[i]$ the edge e in layer i. A new solution γ_u is generated as follows. $\mathbb{C}(\gamma_u)[i] = \mathbb{C}(\gamma_v)[i] + C_{e[i]}$, $W(\gamma_u) = W(\gamma_v)$ and $\mathbb{S}(\gamma_u)[i] = \mathbb{S}(\gamma_v)[i] + \ln 9 \cdot D_{e[i]}$ where $C_{e[i]}$ denotes the capacitance of edge e in layer i and $D_{e[i]}$ denotes the Elmore delay of edge e in layer i for $1 \leq i \leq t$. Thus, the slew characteristics of wire in each layer are maintained in solution propagation. This will allow the layer assignment without backtracking. It is also clear that only uniform wire size is allowed between any consecutive buffers by this operation. It is desired as [12] proves that no wire tapering is needed in prevailing VLSI designs.

"Add Buffer": A buffered solution $\gamma_{u,buf}$ can be generated by inserting a buffer b at u to γ_u as follows. $\mathbb{C}(\gamma_{u,buf})[i] = C_b$, $W(\gamma_{u,buf}) = W(\gamma_v) + W_b$ and $\mathbb{S}(\gamma_{u,buf})[i] = 0$ for $1 \leq i \leq t$. After inserting the buffer, the slew at immediate downstream buffers will be evaluated as $\sqrt{S_{b,out}^2 + \mathbb{S}[1]^2}$ for the default layer $i = 1$ where $S_{b,out}(u)$ denotes the buffer output slew of b at u. In [15], the solution violating the slew constraint will be pruned. It is possible that no buffering solution in the default layer may satisfy slew constraint in presence of a large blockage. In that case, the algorithm in [15] will fail in generating any solution. In contrast, our new buffering algorithm will explore the power of layer assignment to recover slew violations as much as possible.

Layer promotion can be easily implemented. The slew by promoting the wire to the next upper layer is just $\sqrt{S_{b,out}(u)^2 + \mathbb{S}[2]^2}$. This layer promotion process will be repeated until there is at least one solution meeting the slew constraint. If this is not the case even at the thickest routing layer l_t, the solutions with the small slew violations (e.g., slew violations smaller than a threshold T_s) will be selected for propagation. In another word, best possible solutions will still be computed even if the slew target cannot be met. Note that the promoted layer will be recorded in the solution.

"Branch Merge": When two branches are merged into a single branch, the solutions of each branch are also merged. Let Γ_l denote the left-branch solution set and Γ_r the right-branch solution set at the branching node, respectively. A new solution γ' is generated by merging any solution $\gamma_l \in \Gamma_l$ and any solution $\gamma_r \in \Gamma_r$ as follows. $\mathbb{C}(\gamma')[i] = \mathbb{C}(\gamma_l)[i] + \mathbb{C}(\gamma_r)[i]$, $W(\gamma') = W(\gamma_l) + W(\gamma_r)$ and $\mathbb{C}(\gamma')[i] = \max\{\mathbb{S}(\gamma_l)[i], \mathbb{S}(\gamma_r)[i]\}$. Clearly, only wire in the same layer will be merged which is desired as shown in [12].

For acceleration, solution pruning will be performed. Given two solutions γ_1, γ_2 at the same node, γ_1 will be pruned if $\mathbb{C}(\gamma_1)[1] \geq \mathbb{C}(\gamma_2)[1]$, $W(\gamma_1) \geq W(\gamma_2)$ and $\mathbb{S}(\gamma_1)[1] \geq \mathbb{S}(\gamma_2)[1]$. Basically, solutions will be compared by the characteristics of default layer since in general if the capacitance on default layer is smaller, then it is easier to have a buffer solution to fix the slew violation at the thicker layer.

In contrast to [4] which propagates numerous solutions across layers and allows wire tapering, LASR is much more efficient since it almost does not increase the number of solutions compared to slew buffering without layer assignment. Refer to Fig. 8 and Fig. 9 for the pseudo-code of LASR.

5. EXPERIMENTAL RESULTS

We implement LADY and LASR in C++ and embed in IBM physical synthesis tool PDS. Our test-case is extracted from an industrial ASIC chip at 65 nm technology node, where 1x and 2x thickness layers are used. All experiments are run on a 2.3 GHz Power 5 machine under AIX 5.3 operation system.

Table 1 shows the results of LADY algorithm, where X_1 is 10 ps. We apply the delay optimization on 169K nets after detail placement and optimizations on default layer. Existing buffer trees are ripped up and rebuilt. Three algorithms, delay based buffer insertion algorithm without layer assignment (LCL) [4], LADY algorithm (LCL+LADY), and UWS algorithm in [12] (LCL+UWS) are compared on worst slack improvement (the difference between worst slacks before/after), FOM (figure of merit, which is the sum of negative slacks) improvement, algorithm runtime (excluding steiner tree, static timing update and I/O process time), and the usage of wire resources on 2x thickness layer. Note that both LCL and LCL+LADY algorithms use speedup techniques and better memory structures from [18, 16], while LCL+UWS does not use these speedup techniques. Since it is not an apple-to-apple comparison,

we leave out the runtime for LCL+UWS (the real runtime is about 40x slower than LCL+LADY. If with similar speedup techniques, LCL+UWS could be about 4x slower since the algorithm creates more solutions due to its cost function). From the table, it is clear that with layer assignment, optimization can achieve much better worst slack and FOM, and uses less buffers. Also LADY uses uses less buffers and much less wire resources on 2x layer than UWS. It is also interesting to see that LADY algorithm is actually even faster than the baseline algorithm here. It could be the fact that many new solutions generated from using thicker layer will quickly prune the solutions on the default layer when the net is long enough.

Table 1: Comparison of delay based buffer insertion (LCL), simultaneous buffer insertion and layer assignment for delay optimization (LCL+LADY), and simultaneous buffer insertion and uniform wire sizing (LCL+UWS).

	LCL	LCL+LADY	LCL+UWS
Worst Slack Improv. (ns)	1.58	4.24	4.29
FOM Improv. (ns)	14568	27045	27663
Buffers Added	121924	66686	76358
Runtime (seconds)	160	145	
New nets on 2x layer	0	56812	103647

Table 2 shows the results of LASR algorithm. We apply the electrical correction on all nets with slew violations. There are originally 32448 slew violations We compare the slew based buffering algorithm without layer assignment (SB) [15] with LASR (SB+LASR) algorithm on running time and the number of slew violations fixed by the optimization. From the table, we can see clearly that the runtime overhead of LASR is ignorable, but 2x slew violations are corrected with less buffers (almost 97% violations are fixed). The rest of violations may need 4x layer or blockage aware routing.

Table 2: Comparison of slew based buffer insertion (SB) and simultaneous buffer insertion and layer assignment for slew recovery (SB+LASR).

	SB	SB+LASR
# of violations corrected	14731	31108
Buffers Added	81736	74407
Runtime (seconds)	46	58
New nets on 2x layer	0	19924

6. CONCLUSION

Interconnect synthesis becomes increasingly important for the advanced technologies to achieve the high quality design. For 65 nm technologies and beyond, layer assignment, instead of wire sizing, will be a critical part in interconnect synthesis to reduce the interconnect delay, buffer usage and mismatch between synthesis and routing. Uniform layer assignment by assigning each subnet to the same layer will achieve the competitive quality of solution compared to wire/layer taping while keeping lower Via cost. We present several efficient techniques for simultaneous buffer insertion and layer assignment, which has proven to be effective in industrial physical synthesis flow. The complete interconnect synthesis flow, which should also include routing estimation to control layer resources globally, congestion mitigation with layer assignment and buffering, is our current focus where the quality and the efficiency are of high importance.

7. REFERENCES

[1] J. Cong, "An interconnect-centric design flow for nanometer technologies," *Proc. IEEE*, vol. 89, pp. 505–528, April 2001.

[2] P. Saxena, N. Menezes, P. Cocchini, and D. A. Kirkpatrick, "Repeater scaling and its impact on CAD," *IEEE Trans. Computer-Aided Design*, vol. 23, no. 4, pp. 451–463, April 2004.

[3] L. P. P. P. van Ginneken, "Buffer placement in distributed RC-tree network for minimal Elmore delay," in *Proc. IEEE Int. Symp. Circuits Syst.*, 1990, pp. 865–868.

[4] J. Lillis, C. K. Cheng, and T.-T. Y. Lin, "Optimal wire sizing and buffer insertion for low power and a generalized delay model," *IEEE J. Solid-State Circuits*, vol. 31, no. 3, pp. 437–447, March 1996.

[5] C. J. Alpert, A. Devgan, and S. T. Quay, "Buffer insertion with accurate gate and interconnect delay computation," in *Proc. Design Automation Conf.*, 1999, pp. 479–484.

[6] J. Cong and K.-S. Leung, "Optimal wiresizing under Elmore delay model," *IEEE Trans. Computer-Aided Design*, vol. 14, no. 3, pp. 321–336, March 1995.

[7] J. P. Fishburn and C. A. Schevon, "Shaping a distributed-RC line to minimize Elmore delay," *IEEE Trans. Circuits Syst. I*, vol. 42, no. 12, pp. 1020–1022, December 1995.

[8] C. C. N. Chu and D. F. Wong, "Closed form solutions to simultaneous buffer insertion/sizing and wire sizing," *ACM Trans. on Design Automation of Electronic Systems*, vol. 6, no. 3, pp. 343–371, July 2001.

[9] J. Cong, K.-S. Leung, and D. Zhou, "Performance-driven interconnect design based on distributed RC delay model," in *Proc. Design Automation Conf.*, 1993, pp. 606–611.

[10] C.-P. Chen, C. C. N. Chu, and D. F. Wong, "Fast and exact simultaneous gate and wire sizing by lagrangian relaxation," in *Proc. Int. Conf. Computer-Aided Design*, 1998, pp. 617–624.

[11] C. C. N. Chu and D. F. Wong, "An efficient and optimal algorithm for simultaneous buffer and wire sizing," *IEEE Trans. Computer-Aided Design*, vol. 18, no. 9, pp. 1297–1304, 1999.

[12] C. J. Alpert, A. Devgan, J. P. Fishburn, and S. T. Quay, "Interconnect synthesis without wire tapering," *IEEE Trans. Computer-Aided Design*, vol. 20, no. 1, pp. 90–104, January 2001.

[13] W. Shi, Z. Li, and C. J. Alpert, "Complexity analysis and speedup techniques for optimal buffer insertion with minimum cost," in *Proc. Conf. Asia South Pacific Design Automation*, 2004, pp. 609–614.

[14] C. J. Alpert, S. Karandikar, Z. Li, G.-J. Nam, S. T. Quay, H. Ren, C. N. Sze, P. G. Villarrubia, and M. Yildiz, "Techniques for fast physical synthesis," *Proc. IEEE*, vol. 95, no. 3, pp. 573–599, 2007.

[15] S. Hu et al., "Fast algorithms for slew constrained minimum cost buffering," *IEEE Trans. Computer-Aided Design*, vol. 26, no. 11, pp. 2009–2022, November 2007.

[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. Conf. Asia South Pacific Design Automation*, 2005, pp. 13–18.

[17] H. B. Bakoglu, *Circuits, Interconnects, and Packaging for VLSI*. Addison-Wesley Publishing Company, 1990.

[18] W. Shi and Z. Li, "A fast algorithm for optimal buffer insertion," *IEEE Trans. Computer-Aided Design*, vol. 24, no. 6, pp. 879–891, June 2005.