# Layout-Aware Gate Duplication and Buffer Insertion *

D. Bañeres
Universitat Politècnica de Catalunya
Barcelona, Spain

J. Cortadella
Universitat Politècnica de Catalunya
Barcelona, Spain

M. Kishinevsky
Strategic CAD Lab, Intel Corp.
Hillsboro, OR USA

*Abstract*— **An approach for layout-aware interconnect optimization is presented. It is based on the combination of three sub-problems into the same framework: gate duplication, buffer insertion and placement. Different techniques to control the combinatorial explosion are proposed. The experimental results show tangible benefits in delay that endorse the suitability of integrating the three sub-problems in the same framework. The results also corroborate the increasing relevance of interconnect optimization in future semiconductor technologies.**

## I. Introduction

Circuit synthesis is a complex problem that cannot be faced as a whole. For this reason, it is usually decomposed into sub-problems that can be reasonably tackled by algorithms that can provide efficient solutions for each of them. This decomposition leads to solutions far from the optimum when the level of abstraction of some of the sub-problems disregards important details at lower levels.

As minituarization evolves down to deep-submicron technologies, the impact of layout details acquire increasing relevance, since interconnect delays become dominant.

The work presented in this paper combines three different sub-problems in the same framework in such a way that the loss of information between logic and layout synthesis is reduced. The combination is performed by iteratively providing feedback from layout to logic synthesis and vice-versa. The three related sub-problems are: gate duplication, buffer insertion and placement.

*Why these three sub-problems?:* First of all, they are closely related, since they are at the boundaries between logic and layout synthesis. Second, they can be combined with an affordable computational complexity. Incorporating more sub-problems, e.g. technology mapping or routing, would prohibitively increase the complexity.

*How to deal with the combinatorial explosion?:* The combination of placement with other techniques that modify the netlist can be tackled by methods that perform Enginnering Change Orders (ECOs). The approach proposed in this paper takes advantage of these methods to implement the reciprocal feedback between placement and gate duplication and buffer insertion.

Still, there is another source of combinatorial explosion: the potential set of gate/buffer trees that can implement a net with high fanout. To avoid an exponential search of candidates, the fanout points of each net are ordered according to the layout information. The trees are explored/generated using a dynamic programming approach that creates subtrees of adjacent points according to the calculated order. In this way, the set of gate/buffer trees are explored in a similar way as tree-based technology mapping algorithms are executed.

Another important feature of the presented approach is that there are no pre-defined insertion points for the new gates and buffers. In principle, there is total freedom to create any tree. The new gates and buffers are placed on top of the existing layout. Incremental detailed placement is used to legalize the new layout.

*This work was supported by grants from Intel Corporation, CICYT TIN2004-07925-C03-01, European Social Fund, FI and a Distinction for Research by the Generalitat de Catalunya.

## II. Previous work and contributions

Several approaches on buffer insertion [1], [2] and gate duplication [3], [4] have been proposed in the past using a load-based model. Some approaches have also been integrated with technology mapping [5].

Buffer insertion has also been incorporated in the routing step of physical layout. Here, the goal of buffer insertion, also called *repeater insertion*, is to minimize the length and the congestion of the wires among the placed cells. Buffers tend to be inserted on free positions to preserve the legality of the placement. Some of these techniques are based on the dynamic programming approach proposed by Van Ginneken [6], that solves the problem in polynomial time with regard to the number of explored locations. Several extensions to this algorithm have been proposed to improve the runtime, to explore multiple candidates locations for the buffers [7], and to generalize the algorithm to other objective functions, such us power consumption [8]. A Fast buffer insertion technique (FBI) was proposed in [9]. It uses heuristics for predictive pruning and redundancy check. It also supports inverter insertion and sink polarity. It reduces the complexity of the conventional Van Ginneken's approach to $O(nlog^2n)$ with regard ot the number of feasible locations.

Another technique to achieve a buffered routing Steiner tree is a simultaneous construction of a Steiner tree and buffer insertion. This approach is more complex, since it has to deal with the Steiner tree construction NP-hard problem. Many developed methods combine buffer insertion with fast heuristics to compute Steiner trees, ie., A-Tree [10] or P-Tree [11], to reduce the complexity.

In [12], the authors combine A-Trees with Van Ginneken's algorithm. The algorithm builds an A-Tree from sinks to source and performs the buffer insertion from source to sinks with several partial solutions stored in the tree. Another approach is presented in [13], where the buffered Steiner tree is constructed from sinks to source with a combination of LT-Trees [14] and P-Tree with a predefined order of the sinks. This algorithm is optimal depending on the order of the sinks but it has a high runtime complexity due to the explosion of the exploration of feasible locations for the buffers and the construction of P-Trees. Besides, the LT-Tree structure restricts the creation of cascaded buffered trees. In [15], the search space of possible locations is reduced taking into account the obstacles of the layout. Although wire distances between locations are precomputed, this approach still has a high complexity due to the exploration.

Finally, gate duplication has also been used in physical layout design. In [16], an incremental time-driven placement with duplication is proposed. The article introduces the topic of feasible and super-feasible region to place the duplicated node in order to create monotonic critical paths. In the technique presented in [17], a more aggressive duplication is performed. All the cells in the critical path are duplicated and placed into a feasible position using an arborescence tree embedding. Later on, a cell unification operation is done to save area on the global circuit.

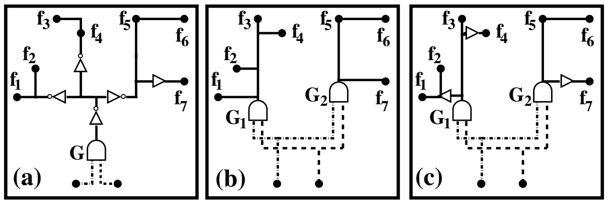

Fig. 1. Example: (a) buffer insertion, (b) duplication, (c) combination of both techniques.

```
BufDup (Net)
{Input: A mapped netlist Net}
{Output: A placed circuit C}
  C := Placement_and_Timing_Analysis (Net);
  do
    Critical_Gates := Calculate_Critical_Gates (C):
    while Critical_Gates $\neq \emptyset \wedge$ cycle time not improved do
      G := Extract_Most_Critical_Gate (Critical_Gates);
      newGates1 := Duplication (G);
      newGates2 := Buffer_Insertion (G);
      newGates3 := Duplication_and_Buffer_Insertion (G);
      NewGates := Select_Best_Solution (newGates1, newGates2, newGates3);
      Insert_Solution_In_Circuit (C, newGates);
      Incremental_Placement_and_Timing_Analysis (C);
      if New_Worst_Slack > Previous_Worst_Slack then
        Undo_Insertion (C, newGates):
      end if
    end while
  while cycle time improved;
  return C;
end;
```

Fig. 2. Algorithm for interconnect optimization.

### A. Contributions

The focus of this paper is the optimization of the interconnection delays taking physical information into account. Buffer insertion and gate duplication are complementary techniques aiming at this goal. An example is shown in Fig. 1(a,b), for a net that connects the source cell $S$ with the fanout cells $\{f_1, \cdots, f_7\}$. Individually, each technique contributes to improve the delay of the net, however the combination of both (Fig. 1(c)) can lead to superior results. The improvement can still be more tangible if physical information is considered and, reciprocally, the changes produced by buffer insertion and gate duplication have a positive impact by incrementally changing the physical layout of the involved cells.

The main contributions of this paper are the following:

- An interconnect optimization approach that combines the exploration of multiple Steiner trees for each net with the incremental placement of the intermediate solutions. In this way, the generated buffers are not restricted to be placed to free spaces. However, a legal placement is still delivered after the optimization.
- The approach integrates gate duplication, buffer insertion and placement in the same framework.
- A gate duplication technique based on a modified layout-aware $k$-means algorithm for clustering [18].
- A dynamic-programming approach to inductively build Steiner trees for buffer insertion. It is based on the approach proposed in [13], with several improvements aiming at (1) the construction of cascaded buffered trees, (2) the smart exploration of feasible locations for the buffers and, (3) the support of gate sizing, inverter insertion and polarity optimization.

## III. Algorithm for interconnect optimization

We present a top-down description of the main algorithm (BufDup) for interconnect optimization. The algorithm is presented in Fig. 2. It receives a mapped netlist as input and produces a placed circuit as output. Initially, cell placement is performed to provide physical information during the interconnect optimization. Delay information is calculated using the Elmore delay model [19]. The timing analysis is performed by considering the physical location of the cells and the Borah-Owens-Irwin (BOI) heuristic for Steiner trees [20]. Although this heuristic is applied individually for each net and does not take into account congestion, it provides a valuable fast lowerbound estimation of the routing cost.

The outermost loop of the algorithm iterates as long as the critical path is improved. At each iteration, the cells at the critical paths are ordered according to their criticality, calculated as a combination of their slack and their fanout. The worst negative slack is the priority factor for optimization, however cells with similar fanout are prioritized according to their higher fanout.

The innermost loop processes gates iteratively according to their criticality. Three different solutions are calculated as shown in Fig. 1: (a) by inserting buffers, (b) by duplicating the gate and, (c) by duplicating the gate and inserting buffers after the duplication. The details on how duplication and buffer insertion solutions are computed will be described in the following sections. Each solution provides a list of new gates to the circuit and has an estimated delay that affects the critical paths of the circuit. The configuration with the best slack time is selected and *physically* inserted in the circuit. Experimentally, we have observed that duplication is mostly selected for gates with high fanout, whereas buffers contribute to reduce the delay on long wires.

The estimated slack time from the new inserted gates does not guarantee the final selection, since the physical location of the new inserted cells may overlap with the existing cells. For this reason, an incremental placement is done to perform slight modifications on the current placement and legalize the position of the new cells. Finally, an incremental timing analysis is performed to check if the selected solution, after legalization, improves the delay. If not improved, the last cell insertion is undone.

## IV. Gate Duplication

Given a gate $G$, gate duplication aims at creating a pair of gates, $G_1$ and $G_2$, such that the original fanout of $G$ is distributed between them. Gate duplication is a well-studied problem [4], [21]. As mentioned in Sect. II, the techniques recently proposed for gate duplication [17] are restricted to legal solutions that do not change the placement of the rest of the cells in the layout. In this section we present a layout-aware gate duplication approach that can be later legalized by incremental changes on the placement.

Clearly, gate duplication explores a trade-off between output and input capacitance. Gates $G_1$ and $G_2$, individually, have a smaller output capacitance than $G$, however the output capacitance of the gates at their fanin increases. The contribution of gate duplication to the performance of a circuit will depend on the particular instance of the problem and the proposed solution.

The algorithm for gate duplication is described in Fig. 3. It is based on the well-known $k$-means clustering algorithm [18]. This strategy is commonly used in data mining where efficient algorithms were proposed to process large quantity of data [22]. The complexity of this algorithm is $O(kni)$, where $k$ is the number of clusters, $n$ is the number of points to be clustered, and $i$ the number of iterations to converge. In our case, $k = 2$ and $n$ is the number of fanouts of the gate, which is typically small. Experimentally, the algorithm converges very fast when $n$ is small, thus showing linear complexity on $n$.

The algorithm aims at clustering the fanout of $G$ into two subsets, one for $G_1$ and another for $G_2$. Initially, two fanout points are

```
Duplication ($G$)
{Input: A gate $G$ to be duplicated}
{Output: Gates $\{G_1, G_2\}$}
   $C_1, C_2$ := Coordinates of two fanouts of $G$;
   while changes in $C_1$ or $C_2$ do
      $S_1$ := {Fanouts of $G$ closer to $C_1$};
      $S_2$ := {Fanouts of $G$ closer to $C_2$};
      $C_1$ := Center of gravity of $S_1$;
      $C_2$ := Center of gravity of $S_2$;
   end while
   $C_{in}$ := Center of gravity of the fanins of $G$;
   Place $G_1$ at the mid-point between $C_{in}$ and $C_1$;
   Place $G_2$ at the mid-point between $C_{in}$ and $C_2$;
   return $\{G_1, G_2\}$;
end;
```

Fig. 3. Gate duplication algorithm.

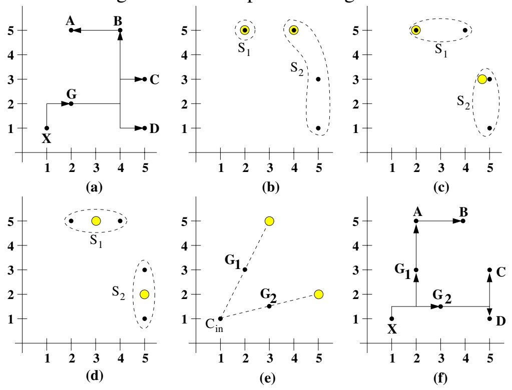

Fig. 4. Gate duplication: (a) Initial net, (b,c,d) evolution of the $k$-means algorithm, (e) calculation of the locations for $G_1$ and $G_2$, (f) possible routing after duplication.

arbitrarily chosen as the potential centers of the clusters and each fanout is assigned to the cluster with the closest center. Iteratively, the centers of each cluster are re-calculated at each iteration as the centers of gravity of the components of each cluster. The calculation stops when a fixpoint is reached.

Figure 4 depicts the evolution of the algorithm. A net with four fanouts driven by gate $G$ is depicted in Fig. 4(a). The gate only has one fanin $X$. Figures 4(b,c,d) show the locations of $C_1$ and $C_2$ (shadowed circles) and the sets $S_1$ and $S_2$ at each iteration[1]. The initial selected points are $A$ and $B$ (Fig. 4(b)), that classify the fanout in two subsets: $S_1 = \{A\}$ and $S_2 = \{B, C, D\}$. After re-clustering, point $B$ is moved to the cluster $S_1$ and convergence is reached.

At the end of the loop, the fanouts are partitioned into the clusters $S_1 = \{A, B\}$ and $S_2 = \{C, D\}$. The location for $G_1$ and $G_2$ is now calculated as the mid-point between the center of gravity of their fanin ($C_{in}$) and the center of the clusters, respectively. In this particular case, $C_{in}$ coincides with the coordinates of the single fanin $X$.

### A. Delay-oriented duplication

The previous method for gate duplication does not take into account any timing information. To amend this unawareness, a postprocess can be performed to re-cluster some nodes before the final location of $G_1$ and $G_2$ is calculated. We next explain the strategy used in our work.

After the clustering algorithm, $G_1$ and $G_2$ may have different criticality according to their slack. Without loss of generality, let us assume that $G_1$ is less critical. Some of the least critical fanouts of $G_2$ that are physically closer to $G_1$ can be *relocated* and assigned to $G_1$. In this way, the total load for $G_2$ is reduced. This process can be iteratively done until the criticality of $G_1$ and $G_2$ is balanced.

[1]To be precise, the figure shows the state of the loop after the calculation of $S_1$ and $S_2$ and before the re-calculation of $C_1$ and $C_2$.

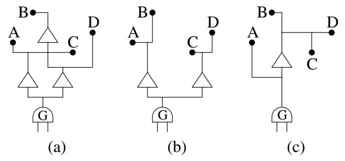

Fig. 5. Buffer trees.

In our approach, we have implemented a greedy postprocess along these lines. The details will not be described in this paper.

### B. Discussion

The current clustering approach is layout-oriented, with a post-process that aims at improving timing by some local re-clustering. One might argue that this could be done the other way around: a time-driven clustering and a layout-oriented postprocess. The initial experiments immediately showed that the chosen approach is superior, since placement has an impact on timing, congestion and routing, which results in better global results after layout.

## V. Buffer insertion

Given a gate $G$ with high fanout, the problem of buffer insertion consists of designing a tree of buffers[2] that drives the fanouts and minimizes the worst negative slack. Figure 5 depicts an example with three different solutions for a gate $G$ with four fanouts.

The number of buffer trees for $n$ fanouts is enumerable but extremely large. To reduce the exploration, we use different strategies.

**Binary trees.** Only binary trees are explored, in which each edge can hold a different number of buffers at different locations, according to their criticality. By only exploring binary trees we are not loosing the chance of building $k$-ary trees. This can be achieved by inserting no buffers at some edges of the tree, as illustrated in the solution depicted in Fig. 5(c), where the buffer is driving three fanouts since one of the sub-trees has no buffers.

**Ordered trees.** The number of possible binary trees with $n$ leaf nodes is[3] $T(n) = n! \cdot C_{2n-1}$. We remove the factor $n!$ by imposing an order in the leaves. In this way, the search is reduced to binary trees whose traversal (pre- or post-order) gives the same order of the leaf nodes. For the examples in Fig. 5, the depicted trees can be represented by the following parenthesized expressions, respectively:

$$((AC)(BD)) \qquad ((AB)(CD)) \qquad (A(B(CD)))$$

Only the last two expressions have the same order at the leaves.

**Which order?** By imposing an order on the leaves, the search space is drastically reduced, but some optimal solutions may be lost. For this reason, it is important to choose good order for the exploration. The order chosen in the proposed approach aims at designing layout-aware trees as follows (see Fig. 6(a)):

> *The polar coordinates (angle and distance) of each fanout with respect to the source node are calculated. The relative position of the nodes is defined by their angle. The distance is used only in the case that the angles are similar. The first and last point in the order is determined by the pair of adjacent fanouts with the largest angle between them.*

[2]We will indistinctively use the term *buffers* to refer to inverting and non-inverting buffers. The optimization of the polarity of the buffers will be briefly discussed at the end of the section V-B.

[3]$C_k$ is the $k$-th Catalan number, $C_k = \frac{1}{k+1}\binom{2k}{k}$, and represents the number of possible binary tree structures with $k$ nodes (a tree with $n$ leaves has $2n-1$ nodes). The factor $n!$ denotes all possible permutations of the leaves.

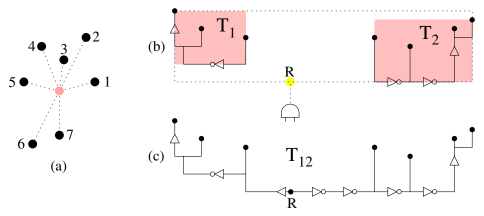

Fig. 6. (a) Order of fanouts, (b) calculation of the root point for two sub-trees and, (c) connection of the sub-trees by repeater insertion.

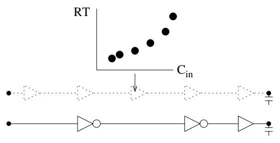

Fig. 7. Repeater insertion.

## A. Bottom-up construction of buffer trees

The exploration of binary trees for buffer insertion is performed bottom-up, from the leaves (fanouts) to the root (gate). This strategy poses a major problem in providing an optimal solution: the criticality of each internal sub-tree of the net is not known until the complete buffer tree has been constructed. For this reason, several solutions are calculated for each sub-tree, each one characterized by a pair $(RT, C_{in})$ that indicates the required time and the input capacitance at the root. The solutions for the left and right sub-trees are combined and provided as the solutions of the whole sub-tree.

In this section, the basic step to construct a tree from two sub-trees is described. This step is illustrated in Fig. 6(b,c), where a tree $T_{12}$ is built from the sub-trees $T_1$ and $T_2$.

The tree is built in two steps:

- Calculate the coordinates of the root $R$. Given the bounding box of the leaves of both sub-trees, the root is the point of the box closest to the source node. In case the source node is inside the box, the root is the source node itself.
- The buffers between the root of the tree and roots of the sub-trees are generated. This is done by a repeater insertion algorithm that is next described.

## B. Repeater insertion

The problem we want to solve is the following: given a library of buffers and inverters and two points (source and sink) with a required time for the sink, design a chain of buffers/inverters that maximize the required time of the source.

This problem is similar to technology mapping for delay and we use the approach presented in [14] for our problem. The approach is simplified and adapted to the design of buffer/inverter chains. The algorithm works in two steps:

1) The number of locations for repeater insertion is calculated. This number is estimated assuming that the same kind of buffer is used along the chain. With this assumption, the potential locations are uniformly distributed along the wire using the optimum number given by Bakoglu's formula [23]

$$N = \left\lfloor \sqrt{\frac{R_w C_w}{R_b C_b}} \right\rfloor$$

where the $R_w$, $C_w$, $R_b$ and $C_b$ are the resistance and capacitance of the wire and the buffer, respectively. The calculation is done using $R_b$ and $C_b$ of the second smallest inverter in the library.

2) A dynamic programming approach for repeater insertion is executed. The algorithm works as a typical delay-oriented technology mapping algorithm [14], from sink to source. For each insertion point, a set of solutions is calculated. Besides the buffers and inverters in the library, the *wire* (no buffer) is also considered as a candidate for mapping.

```
Buffer_Insertion ($G$)
{**Input:** The source gate $G$ (assume the gate has $n$ fanouts)}
{**Output:** A buffered Steiner Tree}
  var:
    $Fanout[1\dots n]$: array of fanouts;
    $Trees[1\dots n, 1\dots n]$: Matrix of buffered trees;

  $Fanout$ := Sort_Fanouts($G$);
  for $f = 2$ to $n$ do
    {Explore sub-trees with $f$ fanouts}
    for each pair $(i,j)$ s.t. $j-i+1=f$, $1 \le i,j \le n$ do
      $R$ := Root node for fanouts $\{i\dots j\}$;
      for $k = i$ to $j-1$ do
        {Create tree from sub-trees with fanouts $\{i\dots k\}$ and $\{k+1\dots j\}$}
        for each pair $(T_1,T_2) \in Trees[i,k] \times Trees[k+1,j]$ do
          $B_1$:=Repeater_Insertion ($R$, Root($T_1$));
          $B_2$:=Repeater_Insertion ($R$, Root($T_2$));
          $T$ := Build_Tree $\left(R \begin{smallmatrix} \nearrow B_1 \rightarrow T_1 \\ \searrow B_2 \rightarrow T_2 \end{smallmatrix}\right)$;

          $Trees[i,j]$ := $Trees[i,j] \cup \{T\}$;
        $Trees[i,j]$ := Select_Subset_of_Best_Solutions ($Trees[i,j]$);

  return Best_Solution ($Trees[1,n]$);
```

Fig. 8. Algorithm of buffer insertion.

Figure 7 illustrates an example of repeater insertion. The dotted chain represents the set of potential points for insertion. At each point, a set of solutions is stored. The chain at the bottom shows a possible solution, in which some of the locations have been simply substituted by wires.

When merging the solutions of the left and right sub-trees, chains with only one buffer at the nearest location of the sink are typically selected when the other sub-tree is critical. This phenomenon is just an algorithmic approach equivalent to the *critical sink isolation* technique proposed in [12].

**Implementation notes:**

- Two sets of solutions are maintained for each chain, one for each polarity. These sets of solutions are propagated towards the root of the tree to deliver the best solution for each possible polarity.
- The exploration of both polarities enables the possibility to handle sinks with negative polarity and to apply *source polarity inversion*. For example, a NAND gate can be substituted by an AND gate (or vice versa) if the complemented polarity of the buffered tree is more convenient.
- To avoid an explosion of solutions, a subset of points is only preserved. The pruning heuristic is explained in Sect. V-D.

## C. Exploration with dynamic programming

The main algorithm is described in Fig. 8. Initially, the order of the fanouts is calculated. The rest of the algorithm calculates the solutions for all possible ordered sub-trees, starting from the smallest trees ($f = 2$) and ending with the complete trees ($f = n$).

Each location of the matrix *Trees* stores several solutions for a sub-tree (only the elements at the upper triangle of the matrix are used). Thus, $Trees[i,j]$ stores all the solutions calculated for the sub-trees with the leaves $Fanout[i\dots j]$. As an example, the sub-trees explored for $n = 5$ are the following:

| | |
|---|---|
| $f = 2$ | (12) (23) (34) (45) |
| $f = 3$ | (1(23)) ((12)3) (2(34)) ((23)4) (3(45)) ((34)5) |
| $f = 4$ | (1(234)) ((12)(34)) ((123)4) (2(345)) ((23)(45)) ((234)5) |
| $f = 5$ | (1(2345)) ((12)(345)) ((123)(45)) ((1234)5) |

For instance, the solutions for the tree $(1(234))$ calculated when $f = 4$ are obtained by combining the fanout 1 with the solutions of the sub-trees with fanouts $\{2,3,4\}$ calculated when $f = 3$, i.e. $(2(34))$ and $((23)4)$.

For every combination of sub-trees, repeaters are inserted from the root of the tree to each root of the left and right sub-trees, respectively. The insertion is done using the approach described in the previous section. The combination of both solutions (`Build_Tree`) also calculates the required time and the input capacitance at the root.

To avoid an explosion of solutions, only a subset of them are kept for each sub-tree. This is performed by the procedure `Select_Subset_Solutions`. The number of solutions has a direct impact on the runtime and the accuracy of the exploration. The strategy for this selection is discussed in the next section.

At the end of the algorithm, the solutions for the complete tree are stored in $Trees[1,n]$. The best solution is returned.

## D. Pruning solutions

During the exploration of solutions for sub-trees and repeaters, several solutions are calculated with different characteristics of required time and input capacitance. To reduce the complexity of the exploration, only a subset of points is selected for further exploration. We next describe an approach that has been proven to be accurate.

First of all, only the Pareto points are represented. If we have $n$ points and we want to select $k < n$, a $k$-means clustering algorithm is executed (the same strategy used on gate duplication [18]), starting with $k$ distributed points along the curve as initial centers. After clustering, the points closest to the centers of the clusters are selected.

## E. Nets with high fanout

The computational complexity can be high for the optimization of nets with very high fanout (e.g. more than 30 fanouts). These cases do not occur very often after fanout optimization, but can drastically delay the execution of a particular instance of the problem.

To alleviate this problem, a pre-clustering strategy is used to partition the fanouts into three clusters, connected to the root node with a buffer. This approach allowed to handle some specific instances with high fanout without any significant loss of performance.

## VI. Experimental results

To validate the approach presented in this paper, three experiments have been conducted: (1) comparison with FBI, (2) results on public benchmarks, and (3) results on future semiconductor technologies.

The $0.13\mu m$ *vxlib ALLIANCE* library [24] has been used for technology mapping. It includes three buffers and four inverters. The technological parameters have been scaled to different technologies using the *Predictive Technology Model* [25]. For 65nm, the wire capacitance and resistance are $2.71\Omega/\mu m$ and $0.19 fF/\mu m$, respectively, that approximately correspond to M2/M3 metal layers of the 65nm technology described in [26].

The experiments have been run on the largest netlists from the ISCAS'99 suite. The initial netlists have been obtained by using the tree-mapping algorithm in SIS, including the fanout optimization step. A square layout with 25% whitespace has been created, with the terminals uniformly distributed around the bounding box.

*Fastplace* [27] has been used to calculate the initial placement. At each iteration, the detailed placer is used for incremental placement. For the final timing analysis, *labyrinth* [28] has been used to estimate the routing trees and calculate the delays using the Elmore model.

| Iter. | fan. | Wire | Initial | FBI | | | BufDup | | |
|---|---|---|---|---|---|---|---|---|---|
| | | Length | Delay | buf | inv | $-\Delta$D | buf | inv | $-\Delta$D |
| 1 | 67 | 6680 | 8531 | 3 | 0 | 901 | 3 | 0 | 1824 |
| 2 | 76 | 6168 | 8246 | 11 | 16 | 1130 | 0 | 3 | 1297 |
| 3 | 17 | 5332 | 7945 | 13 | 0 | 123 | 4 | 1 | 126 |
| 4 | 35 | 6024 | 7870 | 12 | 23 | 394 | 3 | 0 | 313 |
| 5 | 15 | 6313 | 7751 | 6 | 7 | 548 | 6 | 8 | 619 |
| 6 | 28 | 6176 | 7732 | 12 | 10 | 227 | 4 | 2 | 227 |
| 7 | 27 | 5508 | 7692 | 13 | 8 | 202 | 2 | 3 | 202 |
| 8 | 15 | 8871 | 7677 | 13 | 3 | 43 | 6 | 11 | 43 |
| 9 | 11 | 4550 | 7612 | 5 | 1 | 18 | 4 | 9 | 22 |
| 10 | 14 | 5040 | 7590 | 2 | 0 | 81 | 1 | 1 | 89 |
| | | | Tot. | 90 | 68 | 3668 | 30 | 38 | 4762 |

TABLE I
Comparison of FBI and BufDup for individual trees.

## A. Comparison with FBI (Fast Buffer Insertion)

The first experiment compares BufDup with FBI 1.0 with the cost package [9]. To the best of our knowledge, this is the only public domain tool based on Van Ginneken's approach that supports inverter insertion and sink polarity. This experiment has been designed to illustrate the impact of the features of BufDup. The Steiner trees used on this experiment correspond to the output wire of the selected critical gate during the first ten iterations of BufDup on the `b14` netlist. For FBI, the Steiner tree is computed using the *BOI* heuristic [20]. The potential locations for the buffers are the intersection points of the tree. Additional distributed locations have also been included for long wires using the number of buffers defined by Bakoglu's formula [23].

The results are presented in Table I. For each tree, it reports the number of fanouts, the total wirelength of the estimated routing and the initial delay (in *ps*) of the tree using the estimated routing.

For a fair comparison, the BufDup has only been used for buffer insertion (no gate duplication). For each method, the number of buffers, inverters and improvement in delay ($-\Delta D$) are reported.

Only in iteration 4, FBI obtains a better delay reduction than BufDup. In the rest of trees, BufDup obtains a better result or similar. The improvements are significantly better in the first trees (the most critical), with high fanout. The improvements are also tangible in the number of buffers and inverters produced by each method.

The improvements are mainly due to two reasons: (1) the wider exploration of trees in BufDup (binary trees with dynamic programming) and (2) the capability of flipping the polarity of the source gate (see the implementation notes in Sect. V-B).

## B. Academic benchmarks

Table II shows the final results obtained by three methods: (1) FBI, (2) BufDup without gate duplication (label Buf) and (3) BufDup. The parameters of the netlist before buffer insertion are reported in the columns with label `Initial`.Tree-mapping with fanout optimization has been run on these netlists before BufDup. The experiments have been run for a 65nm technology. The last row of the table shows a normalized average of the results. The reported delay corresponds to the one of the critical path in the netlist. Several conclusions can be drawn:

- The layout-aware interconnect optimization reduces delay by 17% with regard to the original netlist after technology mapping and fanout optimization.
- The wide exploration of BufDup (including incremental placement) has a tangible impact in the design of the Steiner trees (delay from 0.91 to 0.83 with regard to FBI).
- The combination of gate duplication with buffer insertion contributes to improve delay in something more than 2% at the expense of 1% area increase.

| Netlist | Gates | | | | Critical-path delay (ps) | | | | Global routing wirelength ($\lambda \cdot 10^{-3}$) | | | | Runtime (sec) | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Initial | FBI | Buf | BufDup | Initial | FBI | Buf | BufDup | Initial | FBI | Buf | BufDup | FBI | Buf | BufDup |
| b14 | 4936 | 5319 | 5128 | 5297 | 8505 | 7763 | 7725 | 7429 | 2828 | 2850 | 2970 | 2996 | 94 | 105 | 195 |
| b14_1 | 4490 | 4616 | 4638 | 4570 | 6018 | 5389 | 5381 | 5184 | 3654 | 2648 | 2731 | 2922 | 50 | 216 | 34 |
| b15 | 7386 | 7906 | 7526 | 7562 | 8102 | 7754 | 7743 | 7657 | 5560 | 5718 | 5915 | 5747 | 48 | 154 | 394 |
| b15_1 | 7189 | 7628 | 7356 | 7448 | 6767 | 6292 | 6135 | 5632 | 4983 | 5214 | 5248 | 5015 | 79 | 114 | 227 |
| b17 | 23358 | 23515 | 23610 | 23603 | 16286 | 14814 | 13831 | 13860 | 26588 | 19943 | 20515 | 20522 | 161 | 487 | 391 |
| b17_1 | 22345 | 23083 | 22709 | 22821 | 6796 | 6156 | 5821 | 5789 | 15167 | 15042 | 15080 | 15022 | 111 | 347 | 518 |
| b20 | 10103 | 10769 | 10417 | 10505 | 10406 | 9592 | 9449 | 8628 | 6645 | 6728 | 6654 | 6606 | 368 | 311 | 357 |
| b20_1 | 8890 | 9037 | 8970 | 8974 | 8403 | 7911 | 7860 | 7948 | 5530 | 5310 | 5443 | 5504 | 51 | 63 | 72 |
| b21 | 10689 | 10971 | 10920 | 10877 | 9513 | 9352 | 9449 | 9156 | 6934 | 6803 | 7352 | 7233 | 98 | 210 | 189 |
| b21_1 | 9005 | 9142 | 9122 | 9110 | 9978 | 9039 | 8581 | 8670 | 9833 | 6527 | 7419 | 7240 | 47 | 107 | 119 |
| b22 | 15233 | 15684 | 15466 | 15591 | 9666 | 8808 | 8536 | 8426 | 9447 | 9756 | 9776 | 9816 | 127 | 265 | 356 |
| b22_1 | 13325 | 13830 | 13542 | 13699 | 9350 | 8531 | 8563 | 8117 | 8094 | 8020 | 7962 | 8089 | 88 | 157 | 306 |
| s35932 | 8432 | 8725 | 8835 | 9010 | 5346 | 4373 | 1912 | 1909 | 5905 | 5944 | 6755 | 6886 | 144 | 250 | 358 |
| s38417 | 10606 | 11008 | 10954 | 10868 | 5310 | 4419 | 3707 | 3750 | 10693 | 7084 | 7743 | 7916 | 99 | 180 | 198 |
| s38584 | 9723 | 9883 | 10137 | 10226 | 5338 | 4804 | 2337 | 2345 | 5258 | 5112 | 6194 | 6137 | 33 | 205 | 267 |
| Norm. | 1.00 | 1.03 | 1.02 | 1.03 | 1.00 | 0.91 | 0.85 | 0.83 | 1.00 | 0.89 | 0.93 | 0.93 | 1.00 | 1.99 | 2.49 |

TABLE II

RESULTS FOR DIFFERENT INTERCONNECT OPTIMIZATION METHODS (65NM TECHNOLOGY).

| Tech. | Gates | | | | Critical-path delay | | | |
|---|---|---|---|---|---|---|---|---|
| | Init. | FBI | Buf | BufDup | Init. | FBI | Buf | BufDup |
| 65nm. | 1.00 | 1.03 | 1.02 | 1.03 | 1.00 | 0.91 | 0.85 | 0.83 |
| 45nm. | 1.00 | 1.04 | 1.03 | 1.02 | 1.00 | 0.87 | 0.82 | 0.81 |
| 32nm. | 1.00 | 1.06 | 1.04 | 1.04 | 1.00 | 0.80 | 0.76 | 0.76 |
| 22nm. | 1.00 | 1.08 | 1.06 | 1.07 | 1.00 | 0.69 | 0.64 | 0.63 |

TABLE III

IMPACT OF INTERCONNECT OPTIMIZATION ON FUTURE GENERATIONS.

On average, the wirelength after global routing is also reduced for FBI and BufDup. The reduction is more important for FBI. Although not reported in the table, *labyrinth* also showed a slight improvement in congestion for all examples. In one case (`b17`), the congestion was reduced by 79% when using BufDup.

### C. Future semiconductor technologies

Table III summarizes the results of BufDup on several future technologies from 65nm to 22nm. The parameters for each technology have been scaled using the interconnection calculator in [25]. The table shows the normalized sums of delay and area of the netlists used in the previous section. The results corroborate that interconnect optimization will acquire an increasing relevance in future technologies due to the dominant role of wire delays. Efficient and accurate buffer insertion approaches will be crucial to reduce critical-path delays. As an example, BufDup was able to reduce the delay by 37% on average for a 22nm technology.

## VII. CONCLUSIONS

An integrated approach for layout-aware interconnect optimization has been presented. The wide exploration of buffer trees using an efficient dynamic programming approach and the incremental legalization of solutions has a tangible impact in the quality of the solutions. The results have also shown the relevant role of interconnect optimization in future technologies.

## REFERENCES

[1] K. Singh and A. Sangiovanni-Vincentelli, "A heuristic algorithm for the fanout problem," in *Proc. ACM/IEEE Design Automation Conference*, June 1990, pp. 357–360.

[2] R. Murgai, "Efficient global fanout optimization algorithms," in *Proc. of Asia and South Pacific Design Automation Conf.*, 2001, pp. 571–576.

[3] C.-H. Chen and C.-Y. Tsui, "Timing optimization of logic network using gate duplication," in *Proc. of Asia and South Pacific Design Automation Conference*, Jan. 1999, pp. 233–236.

[4] A. Srivastava, R. Kastner, C. Chen, and M. Sarrafzadeh, "Timing driven gate duplication," *IEEE Transactions on VLSI Systems*, vol. 12, pp. 42–51, Jan. 2004.

[5] Q. Liu and M. Marek-Sadowska, "Wire length prediction-based technology mapping and fanout optimization," in *International Symposium on Physical Design*, 2005, pp. 145–151.

[6] L. van Ginneken, "Buffer placement in distributed rc-tree networks for minimal elmore delay," in *Proc. International Symposium on Circuits and Systems*, 1990, pp. 865–868.

[7] C. J. Alpert, M. Hrkic, and S. T. Quay, "A fast algorithm for identifying good buffer insertion candidate locations," in *International Symposium on Physical Design*, 2004, pp. 47–52.

[8] J. Lillis, C.Cheng, and T. Y. Lin, "Optimal wire sizing and buffer insertion for low power and a generalized delay model," in *Proc. Int. Conf. Computer-Aided Design (ICCAD)*, 1995, pp. 138–143.

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

[10] J. Cong, K. Leung, and D. Zhou, "Performance-driven interconnect design based on distributed rc delay model," in *Proc. ACM/IEEE Design Automation Conference*, 1993, pp. 606–611.

[11] J. Lillis, C. Cheng, T. Y. Lin, and C. Ho, "New performance driven routing techniques with explicit area/delay tradeoff and simultaneous wire sizing," in *Proc. ACM/IEEE Design Automation Conference*, 1996.

[12] T. Okamoto and J. Cong, "Buffered steiner tree construction with wire sizing for interconnect layout optimization," in *Proc. Int. Conf. Computer-Aided Design (ICCAD)*, 1996, pp. 44–49.

[13] A. H. Salek, J. Lou, and M. Pedram, "A simultaneous routing tree construction and fanout optimization algorithm," in *Proc. Int. Conf. Computer-Aided Design (ICCAD)*, 1998, pp. 625–630.

[14] H. J. Touati, C. W. Moon, R. K. Brayton, and A. Wang, "Performance-oriented technology mapping," in *Proc. 6th M.I.T. Conference on Advanced Research in VLSI*, Apr. 1990, pp. 79–97.

[15] X. Tang, R. Tian, H. Xiang, and D. F. Wong, "A new algorithm for routing tree construction with buffer insertion and wire sizing under obstacle constraints," in *Proc. Int. Conf. Computer-Aided Design (ICCAD)*, 2001, pp. 49–56.

[16] G. Chen and J. Cong, "Simultaneous timing-driven placement and duplication," in *Proc. International Symposium on Field-Programmable Gate Arrays*, 2005, pp. 51–59.

[17] H. Kim, J. Lillis, and M. Hrkic, "Techniques for improved placement-coupled logic replication," in *Proc. Great Lakes Symposium on VLSI*, 2006, pp. 211–216.

[18] J. B. MacQueen, "Some methods for classification and analysis of multivariate observations," in *Proc. of 5th Berkeley Symp. on Mathematical Statistics and Probability*, vol. 1. Berkeley: University of California Press, 1967, pp. 281–297.

[19] W. C. Elmore, "The transient response of damped linear networks," *Journal of Applied Physics*, vol. 19, pp. 55–63, Jan. 1948.

[20] M. Borah, R. M.Owens, and M. Irwin, "An edge-based heuristic for steiner routing," *IEEE Transactions on Computer-Aided Design*, vol. 13, no. 12, pp. 1563–1568, Dec. 1994.

[21] R. Kastner, A. Srivastava, and M. Sarrafzadeh, "On the complexity of gate duplication," *IEEE Transactions on Computer-Aided Design*, vol. 20, no. 9, pp. 1170–1176, Sept. 2001.

[22] T. Kanungo, D. M. Mount, N. S. Netanyahu, C. D. Piatko, R. Silverman, and A. Y. Wu, "An efficient k-means clustering algorithm: Analysis and implementation," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 24, no. 7, pp. 881–892, 2002.

[23] H. B. Bakoglu, *Circuits, Interconnections, and Packaging forl VLSI.* Addison-Wesley, 1990.

[24] Alliance library, www.vlsitechnology.org/html/vx_description.html.

[25] Interconnection calculator, www.eas.asu.edu/~ptm/cgi-bin/interconnect/local.cgi.

[26] P. Bai, C. Auth, and et al., *A 65nm Logic Technology Featuring 35nm Gate Lengths, Enhanced Channel Strain, 8 Cu Interconnect Layers, Low-k ILD and 0.57$\mu m^2$ SRAM Cell*, Intel Developer Forum, Aug. 2005.

[27] N. Viswanathan and C.-N. Chu, "Fastplace: efficient analytical placement using cell shifting, iterative local refinement,and a hybrid net model," *IEEE Transactions on Computer-Aided Design*, vol. 24, no. 5, pp. 722–733, May 2005.

[28] R. Kastner and M. Sarrafzadeh, "Labyrinth: A global router and routing development tool," www.ece.ucsb.edu/~kastner/labyrinth.