A New Incremental Placement Algorithm and its Application to Congestion-Aware Divisor Extraction

Satrajit Chatterjee and Robert Brayton
Department of EECS
University of California at Berkeley
Berkeley, California 94720
Email: {satrajit, brayton}@eecs.berkeley.edu

Abstract—**This paper presents two contributions. The first is an incremental placement algorithm for placement-aware logic synthesis along with a proof of optimality. The algorithm can efficiently compute the optimum location for a newly introduced node in a network that minimizes the incremental increase in the total half-perimeter wire-length of the network. The algorithm can be applied in a variety of placement-aware optimization contexts. The second contribution is a specific application of this algorithm to placement-aware common divisor extraction. We evaluate the effectiveness of the proposed extraction procedure by using it in an otherwise non-placement-aware flow with two different final placers. The first flow uses an industrial congestion-driven placer and results in an average reduction of 21% in congestion as measured by the global router. The second flow uses an academic wire-length-driven placer and results in an average reduction of 11% for a tool-specific measure of congestion estimated from the placement. Our experiments also reveal a rather surprising phenomenon: in many cases the attempt to minimize the wire-length results in fewer literals after extraction than with a conventional literal-driven approach.**

I. Introduction

Wires are playing an increasingly greater role in DSM VLSI design. There are two main reasons for this. First, with gates becoming smaller and faster, designs are being limited by the delays of the wires connecting the gates. This is especially true of the longer wires in the design. Second, the routability of a design after placement is a growing concern in the design community. Both these wire-related issues manifest themselves in the form of synthesis iterations in the design flow. This is because conventional logic synthesis does not begin to consider wire delays until after detailed placement. If timing is not met or if wires cannot be routed, the designer has to go back to the initial logic network and restructure it.

There have been some approaches to alleviate the problem during the later stages of the flow, by effecting what are essentially local changes to the logic during the physical design. This is the approach of "physical synthesis." But at this stage in the flow, it is often a case of too little, too late. It is at the early stages of the flow, especially the technology independent step, that large changes to the network are made; but at this stage usually no wire-information is used.

The present effort aims to explore options during technology independent optimization to address the wiring problems. In particular we are interested in obtaining decompositions which are better from a congestion point of view. We maintain a companion point placement of the network during one of the key steps of the technology independent optimizations, namely common divisor extraction. This is used to guide the extraction process to generate better netlists with shorter wires since empirical evidence shows that minimizing the total wire-length of a design improves the routability. We use the placement information only for this step (and then discard it), and employ conventional algorithms for the rest of the flow. The expectation is that by using the placement to guide the extraction, we obtain a structurally superior netlist, which is inherently easier to route. The exact placement is not very important, but may be viewed as a way to guide the decomposition in an useful manner.

Towards this end, we present a new incremental placement algorithm that is generally useful in placement-aware logic synthesis. The algorithm can optimally place a new node into an existing network so that it minimizes the incremental increase in the total wire-length of the network using the half-perimeter model of wire-length. Since this algorithm is very efficient, it can be used to evaluate candidate options for their contributions to reducing wire-lengths. This allows us to replace earlier heuristic approaches and to obtain superior results with less computational overhead for being placement-aware. While the proposed algorithm is fairly general, we present an application of it to placement-aware extraction for minimizing congestion.

The experimental results are very encouraging. The algorithm succeeds in significantly reducing congestion and is consistent across a variety of benchmarks, across two different process geometries, for different placement engines and for different metrics of congestion. In an industrial flow using a commercial congestion-driven placer, we obtain a 21% average reduction in congestion as measured by the global router. In a different flow, using an academic placer that minimizes wire-length we obtain about 11% average reduction in congestion as estimated by a metric based directly on the placement. In this flow we obtain about a 7% reduction on average in total wire-length after final placement. Note that these congestion reductions are obtained only by altering the extraction procedure to be placement-aware. None of the other steps, including technology mapping are placement-aware. Future work will focus on making the other steps placement-aware.

The rest of this paper is organized as follows. We begin

0-7803-8702-3/04/$20.00 ©2004 IEEE.

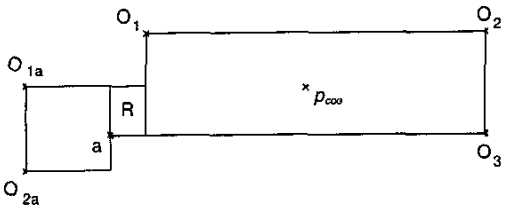

Fig. 1. Example where the center of gravity (p_{COG}) of the fanin (a) and fanouts (o_1, o_2, o_3) of the divisor is significantly different from the optimal location that minimizes the increase in estimated total wire-length (the region marked R). (o_{1a} and o_{2a} are the other fanouts of a.) To see that the p_{COG} is arbitrarily far observe that as the extent of the fanout net increases to the right, the center of gravity moves further to the right away from the optimal region.

by surveying related work in Section II. Next we present an overview of a placement-aware extraction procedure in Section III that motivates the optimum incremental placement algorithm. We develop the algorithm along with a proof of correctness in Section IV. Section V presents experimental results. Finally we conclude in Section VI.

II. Related Work

The idea of maintaining a companion placement while doing logic synthesis was first proposed by Pedram and Bhat in a paper on improving timing closure by taking wire delays into account [9]. They assign a cost to a divisor based on its wire-value. Wire-value is defined as the reduction in total wire-length if that divisor is used. However, this reduction depends upon where the divisor is placed. They note that solving the quadratic optimization problem to determine the optimal location of a divisor is computationally infeasible (since it needs to be done for all candidate divisors, and re-computed for any divisors affected as a result of division). Therefore they settle for an approximation of the optimal location as the center of gravity of the divisor's fanins and fanouts.

Gosti et al. [4] use a wire-planning heuristic based on the placement information to penalize a divisor if there is no "good" placement available for it. A divisor has a good placement if it can be placed without "backtracking." No backtracking is defined as the existence of a placement such that the length of all wires from any fanin of the divisor to any fanout is simply the Manhattan distance from the fanin to the fanout. However since not every good divisor can have such a placement, a divisor is measured by how close it can be placed to avoid backtracking. Also duplication is used in case the backtracking required is too severe. Once a divisor is chosen, it is placed at the center of gravity of its fanins and fanouts.

Kutzschebauch and Stok [7] define the layout cost of a candidate divisor to be the sum of the Manhattan distances between the signal origins of the literals in the divisor and the divisor's fanouts. Among all divisors of roughly the same literal-savings, the divisor having the best layout cost is chosen. The divisor is then placed at the center of gravity of its fanins and fanouts.

We note here that these approaches to incremental placement in the context of placement-aware synthesis do not address the question of optimally locating the nodes in order to accurately quantify their effect on wiring. The center of gravity of the fanins and fanouts of a divisor need not be close to the optimal location that minimizes the increase in total wire-length. Figure 1 provides an example where the center of gravity of fanins and fanouts is arbitrarily far from the optimal location for a divisor. Furthermore the semi-perimeter wire model is a more accurate model of the wire-cost than the Euclidean one. Together, these mean that the extraction procedure based on optimal placement algorithm presented in this paper can more accurately determine and utilize the physical information.

Since the proposed algorithm is linear in the number of nets affected, it can be used to place a selected divisor at the optimum location, as well as to evaluate candidate divisors for their optimal wire-length savings. This makes the heuristic choices used in the previous approaches unnecessary and results in selection of better divisors. Furthermore the placement of divisors has a cumulative effect which results in a quality-runtime tradeoff. If the new divisors are not placed properly (as is the case with the center-of-gravity approaches), subsequent divisor extractions would be working with inaccurate cost estimates which would result in bad choices. This deviation of the estimated placement due to poor incremental placement can be countered by more frequent invocations of the external placer to re-place the entire design but that leads to an increase in run-time.

Note that the results of the layout-aware extraction algorithms in [9] and [7] are presented in the context of complete placement-aware flows [10] and [8] respectively. In contrast the results in the present paper are obtained by using only the layout-aware extraction in an otherwise conventional flow. By making the other steps in the flow also placement-aware the results may be improved further. We stated above that the present approach allows us to obtain a better incremental placement and so fewer calls to the external placer are made to re-place the whole netlist. Thus we expect to have shorter run-times than the other approaches.

Also we note that [4] indicates an increase of about 15% on average in the number of literals, and [7] indicates an increase of about 7% in the literals when layout-aware extraction is used. However, as mentioned earlier in most cases we actually find a reduction in the literals along with a reduction in the final total wire-length after routing.

Connection with structural metrics. There is an interesting connection between the present work and the empirical study of network graph structures presented by Kudva et al. [6]. They found that a graph property called adhesion correlates well with routability. Since in our flow, the placement information is generated and used only during extraction, and then discarded, it would appear that by using the proposed algorithm, we obtain structurally better netlists since congestion is greatly reduced in the final placed design.

III. Placement-Aware Synthesis

A. Overview

The basic idea in placement-aware logic synthesis is to maintain a companion point placement of the network during the technology independent optimizations. The point placement of the network is used to estimate wire-lengths using the half-perimeter of the bounding box of a net.

Technology independent optimization includes operations such as divisor extraction, substitution, elimination, and simplification. These operations change the structure of the netlist. In general terms, this is done by modifying a subset of the nodes in the network, by removing some existing nodes and by introducing new ones. Usually a list of potential modifications is made, and the best modification is chosen. In the conventional flow this best modification is the one that minimizes the number of literals or some other measure of area.

For restructuring the network to minimize the total wire-length, we want to evaluate the reduction in wire-length for all candidate modifications in order to choose the best one. This leads to the need for a quick incremental placement algorithm. We present the algorithm to do this in the next section. In the rest of this section we concentrate on one of the operations, namely extraction, to illustrate the ideas presented above.

B. Conventional Extraction

Common divisor extraction seeks to identify a common subexpression appearing in different nodes in the network and collect it into a single node. Since this significantly changes the structure of the network, it is a natural candidate for placement-aware synthesis. The conventional algorithm *fast extract* [11] generates a list of (single- and double-cube) divisors of the nodes in the network. These divisors are ordered by their literal-gains. Literal gain of a divisor is the reduction in the number of literals in the network if that divisor is used. The divisor with the best literal-gain is chosen, and the division is performed. As a result the gains of some of the other divisors may change. These gains are incrementally updated. Once again, the divisor with the best gain is chosen. The process continues until there are no divisors with positive gain. Note that this algorithm is greedy and not exact.

C. Placement-Aware Extraction

In placement-aware extraction, we modify the divisor selection scheme. Instead of choosing the divisor with the best gain, we want to choose one which also has a good wire-value, i.e. one which reduces the total wire-length in the network. But note that the wire-value of a divisor depends on where it is placed. We would like to place a divisor where it causes the greatest reduction in wire-length and use that to evaluate the gain of the divisor.

Initial Placement. We start with an initial point placement of the network obtained using an external standard cell placer CAPO [1]. Note that for the external placer a point placement does not make sense: As there is no concept of legalization, it can produce a trivial placement where all the nodes are placed on top of each other. To get a non-trivial placement, we assign an area to each of the nodes in the netlist. In our experiments we assign the same area to each node in the network, even though some nodes may have more logic than others. From the placement returned by the placer, we obtain the point placements by assigning to each node the center of its area.

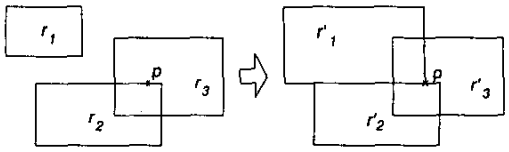

Fig. 2. Expansion of bounding boxes.

It might be argued that a more accurate placement could be obtained by setting the sizes of the nodes to be proportional to the size of the logic function. However if this is done naively, there is a large dynamic range in the cells to be placed leading to a poor placement because of packing issues. Also such a placement is unrealistic, since after mapping, the large nodes are decomposed into groups of smaller cells which are "spread" out in the placement. A more suitable approach might be to break a large node into a group of smaller cells for the placement. The point placement of the large node is obtained as the center of gravity of its constituent cells. Unfortunately, this could lead to a significant increase in runtime since more cells need to be placed initially. The technique outlined in the preceding paragraph may be viewed as an approximation of this scheme.

Divisor Selection. As in the case of the conventional fast extract operation, a list of candidate divisors in created. Instead of ordering them just by their literal-gains, we also want to account for their wire-length gains. We assigned gains to the divisors according to the following formula:

$$\text{gain}(d) = \lambda \cdot \text{wire-gain}(d) + (1 - \lambda) \cdot \text{literal-gain}(d) \qquad (1)$$

where λ is a parameter indicating the relative emphasis on wire-length minimization. (In this context we may view the conventional fast extract as having $\lambda = 0$.)

The wire-gain of a candidate divisor depends on where it is placed. Using the procedure outlined in the next subsection, we place the divisor at the location where the incremental increase in the total wire-length is minimized, and use that to compute the wire-gain. Since the algorithm is very efficient we can afford this computation for all the candidate divisors and assign each divisor its best possible wire-length savings.

Once the best divisor is chosen, division is performed, and all the weights are recalculated incrementally just as in the conventional case. The divisor is placed at an optimum location, and the process repeats. After about 500 divisor extractions, we re-place the entire network using the external placer since the incremental placement may have deviated significantly from the actual placement.

D. Optimal Divisor Placement

The wire-gain of a divisor depends on where it is placed. Observe that the change in the wire-length caused by a divisor extraction can come only from the nets affected by the division. These nets include the fanin nets of the divisor and its fanout net. We explain this with an example.

Example. Suppose that the divisor $d = (a + b)$ is extracted from the node $n_1 = g \cdot (a + b)$ and $n_2 = h \cdot (a + b)$. After the division, there is no need for net a to go to n_1 or to n_2. Thus the bounding box of a may shrink since points are removed from it. However net a must now go to the location of d; hence there could be a possible expansion. Similarly for net b. Furthermore, division introduces a new net for the divisor d into the network which connects the nodes n_1, n_2 and d.

Our task is to place the new divisor d at a location which minimizes the increase in the total wire-length. We compute the bounding boxes (i.e. the rectangles) corresponding to the nets for a and b without considering that these nets also fan out to d. In other words we compute the shrunken bounding boxes. Similarly we compute the (shrunken) bounding box for the newly introduced fanout net of d without considering that this net also includes d. In other words for this output net, we only look at the fanouts of the divisor. When d is introduced, these shrunken rectangles expand to cover d. This is illustrated in Figure 2 where point p is the candidate location for node d. Each rectangle r_i must expand to cover the location p of d. We want to find a location which minimizes the total perimeter of the expanded rectangles.

IV. Optimal Incremental Placement

A. Problem Abstraction

From the discussion above, we have the following abstraction. Given a set of rectangles R (which corresponds to the shrunken bounding boxes of the affected nets), find a point p (corresponding to the new node) such that every rectangle $r_i \in R$ "grows" minimally to a rectangle r_i' to include the point p. More precisely, rectangle r_i' is defined as the bounding box of the 4 corners of r_i and the point p. We have a cost function associated with every location of point p:

$$\text{cost}(p) = \sum r_i' - \sum r_i$$

where we overload r_i to stand for the half-perimeter of rectangle r_i. The objective is to locate p such that cost is minimized. The optimum location for p in general may define a region (see Figure 1). (In the experiments, the new divisor is placed at the center of this region.)

B. Characterizing the Region of Optimality

Lemma 1: $\text{cost}(p) = \text{cost}_x(p_x) + \text{cost}_y(p_y)$ where p_x and p_y are the $X-$ and $Y-$coordinates of p i.e. the cost function is separable.

Proof: For any rectangle r, let $r = r_x + r_y$, where r_x and r_y denote the horizontal and vertical components of the half-perimeter of rectangle r respectively. Therefore,

$$\begin{aligned} \text{cost}(p) &= \sum (r'_{xi} + r'_{y_i}) - \sum (r_{xi} + r_{y_i}) \\ &= \sum (r'_{xi} - r_{xi}) + \sum (r'_{y_i} - r_{y_i}) \end{aligned}$$

Choose $\text{cost}_x(p_x) = \sum (r'_{xi} - r_{xi})$ and $\text{cost}_y(p_y) = \sum (r'_{y_i} - r_{y_i})$ to prove the desired result. ■

Algorithm 1 Compute Optimum Y-Region

INPUT: Sorted list L of rectangle edges (y, t), y is y-coordinate and $t \in \{\text{top}, \text{bottom}\}$
OUTPUT: (y_1, y_2) indicating optimum region and *cost* which is optimal cost of placement

```
i ← 1
m ← length(L)/2
cost ← 0
for each node (y, t) in L in order do
  if i ≤ m and t = bottom then
    cost ← cost − y
  else if i ≥ m + 1 and t = top then
    cost ← cost + y
  end if
  if i = m then
    y1 ← y
  else if i = m + 1 then
    y2 ← y
  end if
end for
return (y1, y2, cost)
```

Lemma 1 allows us to minimize the X- and Y-coordinates independently. Also cost_x depends only on the X-coordinates of the vertical edges. Similarly for cost_y. In what follows we only look at computing the minima for cost_y. In the discussion that follows, note that the $Y-$axis is directed downwards as shown in Figure 3. Let above(y) be the number of rectangles in R, which have the $Y-$coordinates of their bottom edges less than (i.e. above) y. Similarly below(y) is the number of rectangles with their top edges greater than (below) y. If y is not coincident with an edge of a rectangle, then above(y) and below(y) are well-defined. Otherwise, any rectangle whose bottom edge is coincident with y is counted in above(y) and any rectangle whose top edge is coincident with y is counted in below(y). Let $Y = \{y_i\}$ be the collection of $Y-$coordinates of the rectangles in R. The index i is assigned so that the y_is are in increasing order. Y includes both top edges and bottom edges of all rectangles.

Lemma 2: For $y_i < y < y_{i+1}$, $y_i, y_{i+1} \in Y$, $\text{cost}_y(y) - \text{cost}_y(y_i) = (\text{above}(y) - \text{below}(y))(y - y_i)$.

Proof: Observe that the functions above and below are constant in the interval (y_i, y_{i+1}). Now consider the rectangles above y_i. In moving to y, the vertical component of the half-perimeter of each rectangle increases by $(y - y_i)$. Likewise, the vertical component of each rectangle below y_{i+1} decreases by $(y - y_i)$. Also observe that moving from y_i to y doesn't

change the vertical components of the rectangles which span y. From this the result follows. ■

Lemma 3: The region of optimality of $\text{cost}_y(y)$ is contiguous.

Proof: By definition $\text{cost}_y(y) = \sum(r'_{y_i} - r_{y_i})$. Note that each of the functions $(r'_{y_i} - r_{y_i})$ is a convex function, i.e. in the graph of the function, the function always lies below a line segment connecting any two points on the function. Since the sum of convex functions is convex, $\text{cost}_y(y)$ is convex. Therefore the region of optimality is contiguous. ■

Lemma 4: If all elements of Y are distinct and if (y_i, y_{i+k}) is the optimum region then $\text{above}(y_i) = \text{below}(y_{i+k})$.

Proof: Suppose not. Let $k_a = \text{above}(y_i)$ and let $k_b = \text{below}(y_{i+k})$. Suppose $k_a > k_b$. Now clearly, $\text{cost}_y(y_{i-1}) = \text{cost}_y(y_i) + (k_b - k_a)(y_i - y_{i-1})$ since k_a rectangles shrink by an amount equal to $(y_i - y_{i-1})$ and k_b rectangles grow by the same amount.

Since $k_a > k_b$ and $y_{i-1} < y_i$ clearly $\text{cost}_y(y_{i-1}) < \text{cost}_y(y_i)$. But this is a contradiction since y_i is a minimum. Similarly, we can show $k_a < k_b$ also leads to a contradiction. ■

From Lemma 4 it is easy to see that if all elements of Y are distinct then the optimum region must be of the form (y_i, y_{i+1}) since if there was some other edge y in the optimum region it would correspond to either a top edge or a bottom edge of some rectangle (but not both, since all elements of Y are distinct) and hence the optimum region would not satisfy Lemma 4. Now for a point y in the optimum region we can write the cost function as

$$\text{cost}_y(y) = \sum_{N_{\text{above}}} (y - y_i^b) + \sum_{N_{\text{below}}} (y_j^t - y)$$

where y_j^b and y_j^t are the coordinates of the bottom edges and top edges respectively and N_{above} and N_{below} are the sets of rectangles above and below the point y. By Lemma 4, $N_{\text{above}} = N_{\text{below}}$, and the above equation can be re-written as

$$\text{cost}_y(y) = -\sum_{N_{\text{above}}} y_i^b + \sum_{N_{\text{below}}} y_j^t$$

Also from Lemma 4, it is clear that the median coordinates of the set Y must be the end-points of the optimum region. To see this consider any rectangle that spans the optimum region. It contributes one $Y-$coordinate to either side of the optimum region. Every other rectangle must either lie above the optimum region or below it, and there is an equal number of such rectangles. Therefore the optimum region must lie between the median points of Y.

C. Algorithm

We can exploit these properties of the solution to design an efficient algorithm to solve the problem. The algorithm is presented as Algorithm 1.

It receives as input the sorted list L of the $Y-$coordinates provided. Each coordinate is annotated with a tag indicating if it is a top edge or a bottom edge. A single pass is made through L to compute the cost function at the optimum region using the formula presented above i.e. every $Y-$coordinate above the optimum region that corresponds to a bottom edge is subtracted from the cost function and every $Y-$coordinate below the optimum region that corresponds to a top edge is added to the cost function. During the pass, the algorithm also notes the median points which it returns as the optimum region.

There is a subtle point in the theoretical analysis above. From Lemma 4 onwards, we assumed that the points in Y have distinct $Y-$coordinates. But the algorithm presented works correctly even when multiple edges have the same $Y-$coordinates (such as in Figure 3). This is because for theoretical purposes we can assume that edges that coincide are really separated by an infinitesimal distance. Since that would not contribute to the perimeter, the characterization of the optimum region remains valid.

It is clear that the algorithm has a runtime of $\Theta(m)$ in the number of rectangles (assuming that the input coordinates are pre-sorted). Figure 3 shows an example run of the algorithm.

V. Experimental Evaluation

A. Comparison with Fast-Extract

We evaluated the proposed extraction algorithm by comparing it with the conventional fast extract operation in the context of a whole flow from a netlist to a globally-routed layout. We measured congestion after placement and global routing using two different placement engines for final placement. Fairly consistent results were obtained across a variety of benchmarks and two standard cell libraries with different geometries. In this paper we present the data for a 0.25 micron standard cell library with 5-metal layers. We use the larger benchmarks from the ITC99 [2] suite.

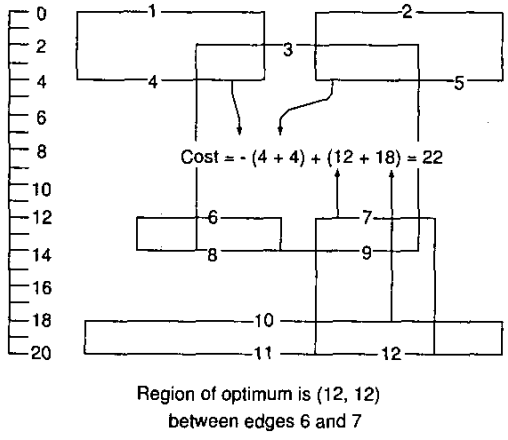

Fig. 3. Example run of algorithm Compute Optimum Y-Region. The number associated with each horizontal edge is the order of that edge's Y-coordinate in the list L. The optimum region in this case is a single point $y = 12$ given by the median edges 6 and 7. All the bottom edges above the optimum region contribute to the cost as do all top edges below the optimum point. Note in particular that edge 7 contributes to the cost.

Benchmark	Estimated Total HPWL after placement			Estimated Congestion after global routing		
	fx	lx-0.5	lx-1.0	fx	lx-0.5	lx-1.0
b14	273471	287264	284730	4.87	3.58	4.61
b15	410375	415254	407789	4.34	3.04	4.02
b17	1447394	1437329	1434812	9.15	7.49	6.46
b20	611470	558594	549182	7.07	5.01	4.15
b21	562556	581372	611775	4.62	5.09	4.63
b22	905092	844017	901761	6.61	5.72	5.26
avg	1.00	0.98	1.00	1.00	0.82	0.79

TABLE I

CONGESTION-DRIVEN FINAL PLACEMENT. COMPARISON BETWEEN PLACEMENT-AWARE EXTRACTION AND CONVENTIONAL EXTRACTION.

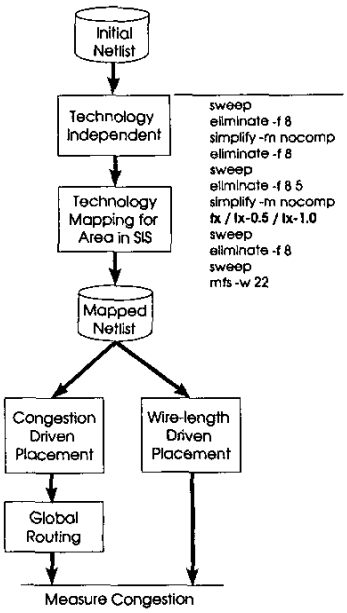

Fig. 4. The design flow used in the experiments described in Section V-A.

Figure 4 shows the flow. Our algorithm is implemented in the MVSIS framework [13]. Starting with the initial network we run a sequence of technology independent optimizations using a script that is similar to the script.rugged script in SIS [12]. We study three different flows for comparison. The first flow, *fx*, uses the conventional version of the fast extract operation. The other two flows – *lx-0.5* and *lx-1.0* – use the proposed placement-aware versions for the fast extract operation. The only difference between the three is the value of λ used in the objective function for choosing the best divisor (Equation (1)). In this context *fx* is like an *lx-0.0*.

Note that even in the placement-aware flows (i.e. $\lambda \neq 0$), no operation except fast extract is placement-aware. In particular observe that in the *lx* flows, the optimizations done after the extraction step do not keep track of placement information, i.e. the actual placement information is effectively lost.

After the technology independent optimizations, the designs are mapped for area into the 0.25 micron standard cell library using SIS. For the final placement we evaluated the performance of the proposed algorithm in two different settings.

Congestion-driven Final Placement. In the first setting we used an industrial congestion-driven placer and global router. The congestion numbers obtained from the global router are shown in Table I. Here, congestion is defined as the percentage of global-router cells where the routing demand exceeds capacity. This metric is an accurate measure of the routability of the design: for this set of place-and-route tools the general rule of thumb is that if a design has more than about 5% over-capacity cells after global routing, the detailed routing is unlikely to succeed. As seen from the table, *lx-0.5* reduces congestion by 18% on average, while *lx-1.0* reduces it by about 21% on average.

The wire-lengths reported in Table I may require some explanation. Although we try to minimize wire-lengths in the technology independent extraction, the placement information is lost, and generated anew during the final placement. The final placement is congestion-driven (and not wire-length driven) and hence moves the cells around so as to minimize congestion. However the placement algorithm can achieve better congestion results in the *lx* flows since the input netlists are more "layout-friendly," i.e. admit placements with inherently less wiring than the *fx* flow.

Wire-length-driven Final Placement. In this case we used the academic placer CAPO [1] for the final placement, and measured congestion from the placement itself rather than a global router. The metric here is the maximum number of nets crossing an edge of the global placement grid. The results are shown in Table II. On average *lx-0.5* reduces congestion by 11% and *lx-1.0* by 9%. In this wire-length driven approach, *lx-0.5* reduces the final placed wire-length by about 7% on average. It is interesting to note that *lx-1.0* does not always reduce wire-length. We believe that this is an artifact of using an unstable placer Capo for the periodic re-placements which injects a certain amount of randomness in the results when

Benchmark	Estimated Total HPWL after placement			Estimated Congestion after placement		
	fx	lx-0.5	lx-1.0	fx	lx-0.5	lx-1.0
b14	243056	238110	240255	40.65	43.54	39.46
b15	343014	338227	350985	53.94	49.37	43.40
b17	1208270	1138170	1195510	54.69	50.86	45.15
b20	521527	443956	515397	42.70	34.65	45.83
b21	510499	453864	492561	39.65	36.26	46.95
b22	786318	740157	802394	57.28	41.29	42.60
avg	1.00	0.93	1.00	1.00	0.89	0.91

TABLE II

WIRE-LENGTH-DRIVEN FINAL PLACEMENT. COMPARISON BETWEEN PLACEMENT-AWARE EXTRACTION AND CONVENTIONAL EXTRACTION.

only wire-length is considered. Of course this can be mitigated by giving some weight to the literal savings, as is the case with *lx-0.5*. Switching to a more stable re-placement algorithm such as Kraftwerk [3] should resolve this problem.

Area. The active cell areas with the different flows are shown in Table III. On average the active cell area is about the same for *fx* and *lx-0.5*, with it increasing slightly for *lx-1.0*. The experiments described in this section were performed with the variable die-model with the utilization set to around 75%. This is done to quantify the congestion improvement purely due to structural improvement. In a fixed die model, the congestion improvement is likely to be greater for the larger benchmarks since the active cell area in the *lx-0.5* flow is actually less than in the conventional flow.

Runtime. The overhead of the incremental placement algorithm is negligible during the decomposition. The runtime is dominated by the calls to the external placement tool, even though that is done only every 500 iterations. Though the actual placement does not take very long to place the technology independent network (order of 30 seconds or less), since the current implementation uses files to communicate, the overhead of I/O and parsing is large. Beyond a better integration of the tools, we hope to obtain better run-times by exploring incremental re-placing options. Our estimates (excluding the I/O) would put the increase due to this algorithm at less than 10% overhead. Indeed the time for the flow as a whole could even reduce, since with an easier design to route, the detailed router may take less time.

Note that the external placement is much faster than a regular placement because we have fewer nodes since the network is not mapped and has fairly large nodes because of the preceding "uphill" eliminate operation; and also because we do not insist on a completely legalized placement.

B. Variation in the Number of Literals

When we varied the emphasis on the wire-length relative to the logic value, i.e. the λ parameter from Equation (1), we observed a rather surprising phenomenon. In many cases (but not all), as we increased the relative emphasis on minimizing the wire-length, the number of literals in the factored form actually decreased slightly. Table IV summarizes the results for the present set of benchmarks. The purely wire-length driven approach actually reduces the literal count by about 2% over the conventional approach.

Though we cannot explain it completely, it would appear that using the placement information results in a better decomposition than using the greedy approach of the conventional fast extract which chooses the divisor that leads to the largest reduction in literal count first. In order to reduce the possibility that this was not due to some random effect in scheduling the divisor extraction, we tested some randomized variants of the fast extract algorithm where instead of choosing the divisor with the highest value, we chose randomly a divisor according to its weight. In these experiments we noticed that almost always the literal count would be worse than in the greedy approach, often by as much as 15%.

VI. CONCLUSION

We presented a new algorithm to incrementally place a new network node such that the increase in total wire-length is minimum. This algorithm should be generally useful in placement-aware logic synthesis. We also presented an application of this algorithm to congestion-aware divisor extraction. Our results are encouraging and consistent across a variety of benchmarks, placement engines and process geometries.

An interesting feature is that significant reductions have

Benchmark	Total Cell Area		
	fx	lx-0.5	lx-1.0
b14	176671	186829	183928
b15	239341	242733	241609
b17	803654	783469	800759
b20	364074	354539	376559
b21	367538	363662	377893
b22	531802	526598	547579
avg	1.00	1.00	1.02

TABLE III

ACTIVE CELL AREA COMPARISON BETWEEN PLACEMENT-AWARE EXTRACTION AND CONVENTIONAL EXTRACTION.

Benchmark	Original after sweep	λ					
		0.0	0.2	0.4	0.6	0.8	1.0
b14	17388	8715	8647	8623	8653	8720	8711
b15	16244	10817	10923	10965	10733	10689	10758
b17	57311	38644	38084	37946	37798	37129	37159
b20	35149	18276	17908	17673	17659	17793	17920
b21	35908	18159	18182	18191	18061	18294	18179
b22	52276	27304	26710	26440	26429	26627	26696
avg		1.00	0.99	0.98	0.98	0.98	0.98

TABLE IV

VARIATION IN THE NUMBER OF LITERALS AFTER TECHNOLOGY INDEPENDENT OPTIMIZATIONS AS A FUNCTION OF λ, THE RELATIVE EMPHASIS IN WIRE-LENGTH MINIMIZATION.

been obtained by modifying only the common divisor extraction operation in an otherwise conventional flow. This may be viewed as further evidence [6] that certain graph structures are inherently better than others with respect to routability. Figure 5 shows the variation in the number of literals and congestion as a function of λ for benchmark b20. The networks for $\lambda = 0.2$ and for $\lambda = 1.0$ have almost the same number of literals, but have significantly different congestion.

A limitation in the current implementation is the use of an unstable placer for the periodic replacement of the network. A more stable placer, or one capable of incremental legalization such as Kraftwerk [3], should improve both run-time and quality of the results further.

Lastly, this method could be used in different settings. For instance, in a flow to minimize delay, we could keep track of the arrival times and slacks in the circuit using a suitable wire-delay model (since the wire-lengths are already known). The proposed algorithm can be used to evaluate divisors on near-critical paths additionally by the wire-length increase they cause on these paths. All other divisors are selected based only on the area saving (though they could still be placed optimally).

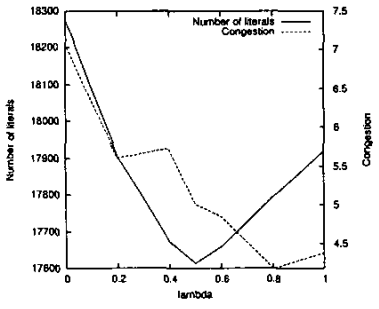

Fig. 5. Variation in number of literals and congestion as a function of λ for benchmark b20.

ACKNOWLEDGMENTS

We thank Alan Mishchenko for his assistance with MVSIS and for his comments on an earlier draft of this paper. This work was supported by C2S2 and our industrial sponsors Cadence, Synplicity, Intel, Magma and Fujitsu.

REFERENCES

[1] A. Caldwell, A. Kahng and I. Markov. "Can Recursive Bisection Produce Routable Placements?," In *Proc. of Design Automation Conf.*, pages 477-482, 2000.

[2] F. Corno, M. Sonza Reorda and G. Squillero. "RT-Level ITC 99 Benchmarks and First ATPG Results," IEEE Design & Test of Computers, Jul-Aug 2000.

[3] H. Eisenmann and F. Johannes. "Generic Global Placement and Floorplanning," In *Proc. of Design Automation Conf.*, pages 269-274, 1998.

[4] W. Gosti et al. "Wire-planning in Logic Synthesis," In *Proc. of ICCAD*, pages 26-33, 1998.

[5] W. Gosti, S. Khatri and A. Sangiovanni-Vincentelli. "Addressing the Timing Closure Problem by Integrating Logic Optimization and Placement," In *Proc. of ICCAD*, pages 224-231, 2001.

[6] P. Kudva, A. Sullivan and W. Dougherty. "Metrics for Structural Logic Synthesis," In *Proc. of ICCAD*, pages 551-556, 2002

[7] T. Kutzschebauch and L. Stok. "Layout Driven Decomposition with Congestion Consideration," In *Proc. of DATE*, pages 672-676, 2002.

[8] T. Kutzschebauch and L. Stok. "Congestion Aware Layout Driven Logic Synthesis," In *Proc. of ICCAD*, pages 216-223, 2001.

[9] M. Pedram and N. Bhat. "Layout-driven Logic Restructuring and Decomposition," In *Proc. of ICCAD*, pages 134-137, 1991.

[10] M. Pedram and N. Bhat. "Layout Driven Technology Mapping," In *Proc. of Design Automation Conf.*, pages 99-105, 1991.

[11] J. Rajski and J. Vasudevamurthy. "The Testability-Preserving Concurrent Decomposition and Factorization of Boolean Expressions," *IEEE TRANS. on Comp. Aided Design*, vol. 2, no. 6, pages 778-793, 1992.

[12] E. Sentovich et al. "SIS: A System for Sequential Circuit Synthesis," Technical Report UCB/ERL M92/41, Univ. of CA at Berkeley, May 1992

[13] Multi-Valued Logic Synthesis System. On the Internet: http://www-cad.eecs.berkeley.edu/Respep/Research/mvsis/