Congestion Aware Layout Driven Logic Synthesis

Thomas Kutzschebauch
Leon Stok
IBM TJ Watson Research Center
Yorktown Heights, NY
{kutzsche, stokl}@watson.ibm.com

Abstract

In this paper, we present novel algorithms that effectively combine physical layout and early logic synthesis to improve overall design quality. In addition, we employ partitioning and clustering algorithms to achieve faster turn around times.

With the increasing complexity of designs, the traditional separation of logic and physical design leads to sub-optimal results as the cost functions employed during logic synthesis do not accurately represent physical design information. While this problem has been addressed extensively, the existing solutions apply only simple synthesis transforms during physical layout and are generally unable to reverse decisions made during logic minimization and technology mapping, that have a major negative impact on circuit structure.

In our novel approach, we propose congestion aware algorithms for layout driven decomposition and technology mapping, two of the steps that affect congestion the most during logic synthesis, to effectively decrease wire length and improve congestion. In addition, to improve design turn-around-time and handle large designs, we present an approach in which synthesis partitioning and placement clustering co-exist, reflecting the different characteristics of logical and physical domain.

1 Introduction

The increasing complexity of microelectronic designs and the continuous development of smaller sized fabrication processes create new challenges to existing design automation tools. One of the most important problems in the design of VLSI circuits is the interaction of logical and physical domains. Due to the development of ever finer-featured integrated circuits, the traditional separation of logic and physical design proves to be disadvantageous as the cost functions employed during logic synthesis become increasingly inaccurate during later design stages.

While this problem has been addressed extensively, all existing solutions employ only trivial logic synthesis transforms late in the design process, during physical design [11, 12]**.** These mainly local transformations are generally unable to reverse adverse decisions made during logic minimization and technology mapping based on inaccurate cost functions, namely literal count and gate-based path delay, not reflecting the physical properties of the design. Existing layout driven logic synthesis algorithms [9] only utilize the layout information for improved timing information using estimated wire delay and to minimize area using cell and estimated wiring area. However, reducing overall congestion for better wireability, a very important objective function besides timing closure, in conjunction with timing and area optimization, has not been addressed. In [10], Pedram et. al. propose a decomposition and factoring based on a relatively simple one-dimensional input location assignment. We present an exact and a greedy algorithm to perform decomposition based on an accurate two-dimensional input location modeling.

In addition, the increasing size of designs leads to infeasible turn-around times. This particularly applies to the field of logic synthesis. Logic minimization algorithms, e.g. kernel extraction and factoring, among others, are generally expensive algorithms. Therefore, current design sizes already present a considerable challenge to existing tools, while future synthesis tools are expected to handle millions of gates in a reasonable amount of time. Existing solutions often require the designer to manually partition a circuit which results in a much more complex design flow and leads to sub-optimal results during placement. Effective approaches using a multi-level clustering have been proposed in [1, 2, 3]. However, these existing approaches only reflect connectivity properties and not the functionality of the netlist. To incorporate clustering into a layout driven synthesis flow, simultaneously existing synthesis partitioning and placement clustering is a necessity.

To solve the aforementioned problems, we have developed algorithms employing physical design information early during logic independent optimization. Specifically, we create an initial placement of the technology-independent netlist and apply the thereby gathered layout information during logic optimization and technology mapping. We utilize the geographical location of the objects in the netlist, for example the location of a signals origin, and the estimated wire length to perform

congestion aware decomposition and technology mapping. By decomposing gates depending on the location of their sources and grouping gates that are located close to each other during technology mapping, we are able to reduce total wire length and overall congestion by distributing the wiring instead of concentrating it in a small area. Furthermore, to handle the increasing complexity of designs, we use a partitioning algorithm for synthesis that identifies and groups reconvergent regions using signal flow which extends the previous work in [8]. We use a simple connectivity-based clustering to speed-up the placement. In addition, clustering for placement generally improves overall congestion as high connectivity is grouped in clusters and more evenly distributed over the entire layout area.

The main contributions of this paper are congestion aware decomposition and technology mapping algorithms using geographical information coupled with effective coexisting synthesis partitioning and placement clustering to form an integrated synthesis and placement flow.

The remainder of this paper is organized as follows. In section 2 we explain the general concept of layout aware logic synthesis and propose our layout based decomposition and technology mapping algorithms. Section 3 describes our synthesis partitioning algorithm based on reconvergent regions and explains the different nature of partitioning for synthesis and clustering for placement. In section 4 we present our integrated synthesis and placement flow, while section 5 shows experiments and results, followed by conclusions and future work in section 6.

2 Congestion Aware Synthesis

To effectively combine logic synthesis and placement, we create an initial placement of the technology-independent netlist and use the placement coordinates of the objects to improve synthesis transformations. For this purpose, we use a quadratic placement solver with quadrisection to minimize quadratic net length. This type of placement algorithm is relatively stable with respect to local netlist changes. Even though this placement is not legal and the netlist is technology independent, it represents a good estimation of the actual location of the objects in the final design.

Decomposition and technology mapping are among the most important steps determining overall congestion and wire length, as we break up technology-independent gates and group them into their technology gate implementation. Following, we will outline our congestion aware decomposition and technology mapping algorithms.

2.1 Decomposition

Our decomposition algorithm first groups the netlist into large gates with many inputs and then decomposes the netlist into 2-input gates. It works on a network consisting of only AND and XOR primitive cells with input and output negations. In the first step, we reduce the number of levels in the design by moving a fanout stem from the output of a gate to its inputs. Following, we combine adjacent gates of similar functionality. In the final step, the circuit is transformed into a fanout-free decomposition consisting of 2-input gates.

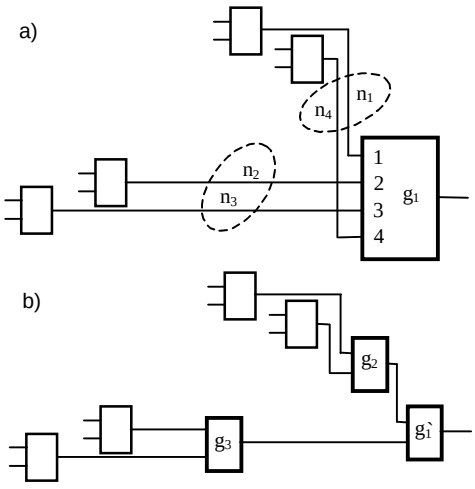

Figure 1: Layout based Decomposition Example

Motivation: Traditionally, the algorithm [14] uses the arrival time at the input pins to create a delay optimal decomposition. In our layout aware approach, we include wiring delay based on the estimated net length and geographical location of the input signals. Let us first demonstrate the importance of geographical location for decomposition on a simple example. Consider the case of decomposing a gate with four input pins and respective arrival times $t_{AT,1} \ldots t_{AT,4}$, as shown in Figure 1a. Assume the order of arrival times to be $t_{AT,1} \le t_{AT,2} \le t_{AT,3} \le t_{AT,4}$, however, all arrival times are within close range, i.e. the slowest signal is only marginally slower than the fastest signal, or identical. A timing based decomposition algorithm would pick the first two fastest pins, i.e. pin p_1 and p_2 of gate g_1 and group them into a 2-input gate. Following, pins p_3 and p_4, now the fastest pins will be decomposed. It is obvious that due to the geographical location of the feeding signals, this will not only cause longer net length, but also higher wiring congestion. Consequently, the better choice is to decompose pins p_1 and p_4 and their respective nets n_1 and n_4 into gate g_2, and nets n_2 and n_3 feeding pins p_2 and p_3 into gate g_3, as shown in Figure 1b, effectively resulting in shorter wiring length and better congestion. In conclusion, we comprehend that it is important to group input pins into a gate to be extracted those signal origins are close to each other. Accurate decomposition is particularly important as any decision made at this point is in most cases only reversible at significant cost in later design stages.

In [10], Pedram et. al. propose a layout based decomposition and factoring algorithm which uses a relatively simple assignment of the input signal locations. All incoming signal edges are placed along a circle and the best combination of nearby signals along the circle is selected. However, this approach only considers one dimension, in fact, signals could be nearby, yet placed on opposite sides of the circle. Hence, a two-dimensional approach is needed.

Problem Formulation: Based on the previous observations, we can formulate the problem of decomposing a subject graph into a network of 2-input gates as follows. We define the cost of grouping two candidate pins into a gate to be decomposed as the Manhattan distance between their respective signal origins. Following, we want to minimize the total cost of grouping subject to a timing constraint. That is, to combine layout information with timing, we only consider candidate pins within a certain timing range. Therefore, the problem can be formulated as follows:

$$Min(\sum_i L_1(p_{1,i}, p_{2,i}))$$
$$s.t. \quad Max(t_{AT,1,i}, t_{AT,2,i}) < T_{\max,i} \tag{1}$$

In the above equation, $L_1(p_{1,i}, p_{2,i})$ is the Manhattan distance between the signal origins of candidate pins p_1 and p_2 in iteration i of the decomposition process. Only candidate pins with an arrival time t_{AT} below an upper bound T_{max} are considered. To find the optimal decomposition minimizing our cost function, we have to consider at most $n \cdot (n-1)/2$ possible decompositions during the first iteration, where n is the number of input pins of the original target gate to be decomposed. Each decomposition of one 2-input gate removes two candidate pins but introduces one new candidate pin, thus changing the input space of the problem. Hence, in the next iteration, we have at most $(n-1) \cdot (n-2)/2$ possible decompositions. The total problem complexity follows as $n! \cdot (n-1)!/2^{n-1}$.

An Exact Algorithm: To reduce the complexity of the problem, we employ a branch and bound algorithm. We traverse the search space depth first choosing the path with minimum cost. Our upper bound is the best solution found thus far. Since our objective function subject to minimization is monotonous, branches yielding a cost higher than the upper bound can be pruned. In addition, the search space is generally smaller since only candidate pins within a certain arrival time range are considered. However, the algorithm is still prohibitively expensive for large n and a less expensive approach is needed.

A Greedy Algorithm: Our greedy approach chooses the best possible solution during each iteration of the decomposition process. Thus, we select the pair of candidate pins with the minimum distance between their respective input signal origins. Obviously, the greedy approach might yield sub-optimal results as the best solution within an iteration might force suboptimal input

```
Order candidate pins by arrival time;
for (i=1 to n)
  for(j=i+1 to n) {
    Calculate L1 distance(pin i, pin j);
  }
}

Greedy_Decomposition(gate g) {
  For_all_pairs_of_pins(t_AT < t_AT,min + T)
    (p1,p2) = Pins with min L1 distance;
  Decompose(g, p1, p2);
  If (pins of g == 2) return;
  Remove p2 from candidate pins;
  Update arrival time of p1;
  Update L1 distances(p1);
  Greedy_Decomposition(g);
}
```

Figure 2: Greedy Decomposition Algorithm

pin pairing during later iterations. However, particularly for large n, the input space is large enough to yield near-optimal results at a much lower complexity compared to the exact algorithm. The pseudo code of our greedy algorithm is shown in Figure 2. We first order the input pins by increasing arrival time t_{AT}, considering gate delays and estimated wiring delay and calculate the Manhattan distances between all pairs of pins. The wiring delay is estimated assuming one-bend connections. Note that calculating distances for all pairs of candidate pins is of quadratic time complexity, however, the candidate set is a function of the maximum number of inputs of the decomposed gate, and not related to the size of the design. We then choose candidate pins using a timing window of a defined width T, hence we only consider pins with an arrival time faster than the minimum arrival time of all pins plus the width of our timing window T. Typically, we choose the value of T as the delay of one 2-input gate, i.e. one level of the decomposition tree. Following, we select the pair of pins (p_1, p_2) with the minimum cost of grouping, i.e. the minimum distance between the locations of their signal origins. After the actual decomposition, pin p_2 is removed from the set of candidate pins and pin p_1 is substituted with the newly created pin. Arrival times of p_1 and L_1 distances to all other pins are updated. The process stops when the original target gate is completely decomposed into a tree of 2-input gates.

In other words, we choose the pair of pins based on a combination of fast arrival time and closeness of its sources in the physical layout. The selection process in the

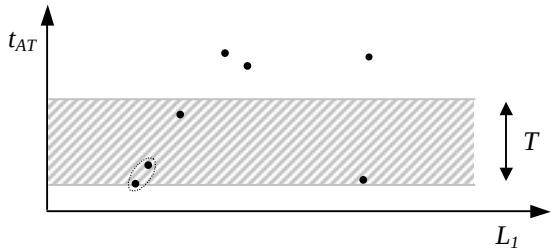

Figure 3: Timing Window based Set of Candidate Pins

three-dimensional space (x, y, t) is illustrated in Figure 3. The timing window of size T is shown as shaded region while candidate pins are shown as dots. For simplicity, the two-dimensional coordinate space (x, y) is pictured in one dimension as the manhattan distance L_1 between the signal origin of a given pin i and all other pins $j \neq i$, hence showing only one plane of the entire solution space.

Each newly created gate in the decomposition process has to be assigned to a placement coordinate. We calculate the new location of the gate as the center of gravity of all input and output connections of the new gate to existing objects with a given location. Therefore, we maintain layout locations throughout the entire decomposition process for all objects in the design.

In general, by ignoring the arrival time at the input pins, the algorithm performs purely layout based decomposition, e.g. grouping pins based only on geographical location. In that case, the set of candidate pins includes all input pins of the gate to be decomposed.

2.2 Technology Mapping

Following decomposition, we technology map the netlist using layout information in a similar manner. Our technology mapping algorithm is based on the concept of wavefront technology mapping as described in [14]. The wavefront algorithm traverses the network in a levelized order from the primary inputs to the outputs using a wavefront of specified width w. Within the width of the wavefront, bound by its *head* and *tail*, all possible technology matches are created and implemented, effectively creating multi-source nets. The final technology implementation is chosen by identifying the fastest driver pin of the multi-source net. Due to the load-independent delay modeling in our gain-based delay model, a delay optimal technology implementation is created. We refer to [14] for a complete description of wavefront technology mapping under a gain-based delay model.

Let us illustrate on an example how layout information benefits technology mapping. Consider the example circuit in Figure 4a. We assume a wavefront width of 3 levels, as shown by the dashed lines. Within the width of the wavefront, all technology matches will be created. In our example, we create the following matches, among others. Technology-independent gates g_1, g_2, g_3, g_5 and g_6 can be implemented with an AO222 and g_4 as an AND gate, or gates g_1, g_2, g_3 as AO22, g_4 and g_5 as 3-way AND and g_6 as an OR gate. Only considering timing and/or area, the first implementation, shown in Figure 4b might be chosen. However, in our example, gates (g_1, g_2, g_3), (g_4, g_5) and (g_6) are placed in different regions of the circuit, i.e. their locations are distant from one another. Therefore, grouping gates (g_1, g_2, g_3, g_5, g_6) will result in increased wiring and local congestion because their input sources are located in different regions of the circuit, which is exactly the reason why these gates are placed far away from each other. If

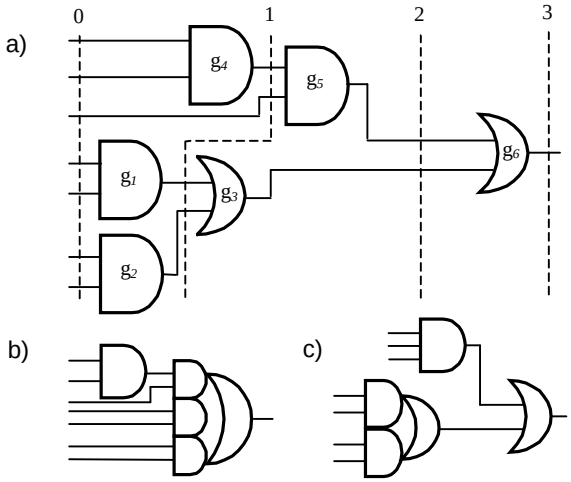

Figure 4: a) Original circuit, b) Generic Mapping, c) Layout driven Mapping

instead, we group gates (g_1, g_2, g_3), (g_4, g_5) and (g_6), as shown in Figure 4c, we not only decrease total wire length because fewer wires are used along the long connections, we effectively improve overall congestion. In Figure 4b, wiring congestion is created because wiring is concentrated by the big AO222 gate having many long wires. Contrary, we have fewer long wires and a better distribution of wiring in Figure 4c. Similarly to decomposition, we define a cost function that weighs timing (and/or area) and layout information. To effectively group close objects, we define a cost of grouping by calculating the additional wiring created by moving each technology-independent gate from its existing location to the new location, i.e. the placement location of the actual implemented technology gate. The location of the technology gate is determined by calculating its center of gravity with respect to all input and output connections.

We define the cost of grouping n gates as follows:

$$C_g = \sum_i \left(\sum_{j \in P(i)} \left(L_1(s_{i,j}, g_i) - L_1(s_{i,j}, m) \right) - w_i \right) \qquad (2)$$

In the above equation, $L_1(s_{i,j}, g_i)$ is the Manhattan distance between $s_{i,j}$, the external source or sinks of pin j in the technology independent gate i and $L_1(s_{i,j}, m)$ is the distance between the same source and sinks and the mapped gate m, while w_i is the wiring saved by eliminating the internal wires. Note that calculating the additional wiring in certain cases creates a small error when using the location of objects whose final technology implementation, and hence final location has not been determined yet. This error is generally small as it is offset by the fact that we only group close objects, thus the location of the final technology implementation is close to the location of the technology-independent objects it implements. However, it can be improved by using a two-pass approach. Instead of keeping only one match, two or more possible matches are

stored. During the second pass, the best combination of matches can be chosen using the known location of the technology implementation.

In conclusion, instead of choosing the technology implementation with the minimal arrival time $t_{AT,m}$ (and/or minimal area), we choose the technology dependent implementation as the minimum of the cost function $\boldsymbol{C_g}(t_{AT}, x, y)$ subject to a timing (or area) constraint. Similarly to our cost function for decomposition, we consider estimated wiring delay and choose a timing window to consider only technology matches within a certain range of arrival times. However, by eliminating timing, the algorithm becomes completely layout based and will choose matches solely based on the geographical information.

In conclusion, by incorporating geographical layout information into the cost function for technology mapping, we are able to reduce total wire length resulting in less global congestion. Even more important, by grouping gates that are located close to each other in the layout, we distribute the wiring instead of concentrating it in single points which yields less local congestion.

3 Partitioning and Clustering

To handle the increasing size of designs, we apply partitioning and clustering to reduce overall problem size. This allows us to handle large designs and achieves a faster overall design turn around time. In addition, our layout aware decomposition and mapping algorithm rely on an initial placement of the technology-independent netlist. In most cases, the number of objects in the technology-independent representation will be significantly larger than the final technology implementation, thus, clustering of the netlist to reduce the number of objects is a necessity.

An important aspect of clustering and partitioning in the overall design flow is the decidedly different nature of logical and physical domain. The objective functions of partitioning a netlist for logic optimization and physical design are substantially different. The main goal during placement is to minimize the total netlength of a weighted graph, thus clustering algorithms focus on grouping highly connected objects, based on local [4] as well as global connectivity [5]. A clustering approach, however, is generally not suited for logic synthesis. Instead of clustering objects which reduces the overall problem size by reducing the number of objects, the obvious solution is to partition the netlist into parts of disjoint functionality. In most cases, complete disjointness does not exist, however, minimizing the sharing of common boolean expressions between the partitions is an excellent objective function to allow the most freedom of choice for restructuring and technology mapping algorithms within each partition.

In order to integrate synthesis and placement, we need to combine the different clustering and partitioning approaches reflecting the distinct nature of logical and physical domain. Therefore, we propose an approach with simultaneously existing partitions for synthesis and clusters for placement. Effectively, we have two different representations for each domain that exist simultaneously in the overall flow. Hence, each object is part of two different representations of the netlist, i.e. a gate is part of a placement cluster as well as part of a synthesis partition.

For placement, we group the netlist into fine-grained clusters based on local connectivity. Clustering also helps to improve congestion as high connectivity is grouped in clusters and more evenly distributed over the entire layout area. Our cluster size is fairly small, which is necessary to still provide enough detail to accurately determine the geographical locations of individual objects which provides the input to our layout driven decomposition and technology mapping algorithms as described in the previous section. Our synthesis partitioning is based on the analysis of reconvergent signal regions which has been applied to test generation and fault simulation [6, 7], and more recently to circuit partitioning for resynthesis of large networks [8]. For easier understanding, we revisit the following known properties of a circuit graph $G(V, E)$.

Definition 1: *Given nodes r and s $\in$ V, if there are more than one disjoint path from s to r, r is a* ***reconvergent node*** *for s, and s is a* ***reconvergent fanout stem****.*

Definition 2: *Let s be a reconvergent fanout stem. Then, a* ***reconvergent region*** *of s consists of all the nodes and output edges that are located on all the paths from s to any of its final reconvergent nodes r that do not drive any other reconvergent node of s.*

We use a modified version of the algorithm in [7] to find all reconvergent regions of a node *s* by searching for all reconvergent nodes of a fanout stem. Unfortunately, this algorithm is of complexity $O(n^2)$, however, we noticed that the size of a reconvergent region is limited, and usually independent of the number of total nodes, i.e. the size of the design. Therefore, we limit the search space from a fanout node to a constant k and achieve linear time complexity $O(n)$ which is much better suited for partitioning large designs.

To create the final partitions, reconvergent regions are

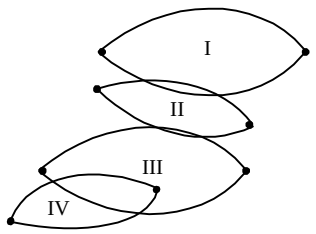

Figure 5: Partitioning of Reconvergent Regions

grouped based on region overlap. Contrary to the greedy approach used in [8], we group reconvergent regions in the order of decreasing overlap and favor regions that overlap within at least one of the final reconvergent nodes of either region and at least one other input node, also known as converging regions. Consequently, we group regions with the maximal number of common boolean expressions to improve synthesis quality. The overlap of two regions is easily computed during the backward phase of our modified algorithm of [7] that identifies the reconvergent regions. Again, this algorithm is of linear time complexity as the number of possible memberships of a node is limited by the constant search space, i.e. the limited size of a reconvergent region.

In many cases, almost all regions are overlapping, thus grouping all regions would result in covering almost the entire circuit in only one or two partitions. Therefore, we use an upper bound to control the maximum size of a partition. Hence some regions can initially not be included in any of the partitions because they overlap more than one existing partition and merging the partitions would result in a too large partition. Their disjoint part is added to the partition with the most overlap. After grouping all regions into partitions, all remaining nodes which are not part of a reconvergent region are absorbed into their neighboring partitions based on their connectivity to input lines of these partitions.

Consider the example shown in Figure 5. Initially, we merge regions III and IV due to the maximum number of shared nodes, creating partition P_1. Note that region IV converges with region III. Now, regions I and II have the most overlap, however, region II is also shared with the newly created partition P_1. Dependent on the maximum partition size allowed, we either add regions I and II into P_1, or merge region I and the non-disjoint part of region II to form a second partition P_2.

After partitioning the design, we perform logic minimization and technology mapping on each of the individual partitions. By dividing the design , we achieve a speedup because most expensive synthesis transforms have a time complexity far greater than O(n). For optimal speedup, the partitions would have to be synthesized in parallel. However, to take full advantage of parallelism, dynamic timing assertion generation at the partition boundaries is necessary. This is generally a non-trivial task that will be address in future work.

4 Integrated Synthesis and Placement

Following, we explain how the concepts outlined in the previous sections are combined to form an integrated synthesis and placement flow. The overall flow is depicted in Figure 6. We start with the technology independent, unoptimized netlist of the design. Initially, we partition the design using our synthesis partitioning algorithm based on reconvergent signal regions, as explained in the previous section. In the next step, we run logic minimization algorithms, e.g. kernel factoring, common subexpression extraction, on the synthesis partitions. Subsequently, we cluster the netlist using our connectivity based clustering algorithm into relatively small clusters. In our model, both cluster and partition based representations coexist to serve the individual needs of logic synthesis and placement. In other words, pure synthesis operations only utilize the synthesis partitions while placement only uses the cluster representation. The layout-driven part of logic synthesis however, i.e. our previously proposed decomposition and technology mapping algorithms make use of both representations.

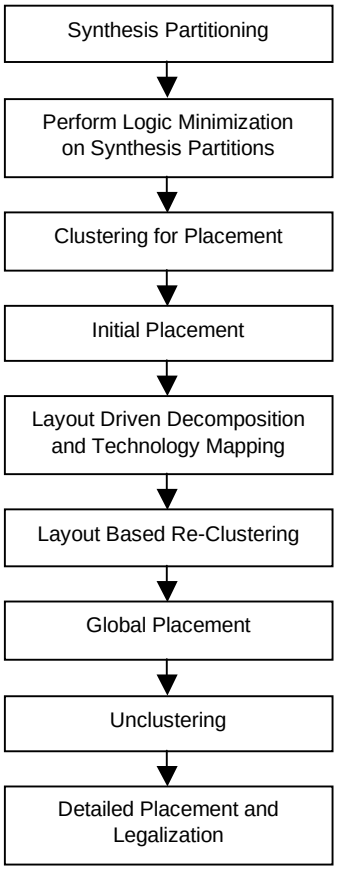

Figure 6: Integrated Synthesis and Placement Flow

After clustering, we create an initial placement using a quadratic placement algorithm with quadrisection to place the clustered technology-independent netlist. Each member of a cluster is assigned the layout location of the parent cluster. Succeeding, we perform congestion-aware layout driven decomposition and technology mapping, as described in the previous sections, on the synthesis partitions. Note that we retain placement information by assigning a layout location to each newly created gate. However, retaining cluster information at this point is an infeasible task. Instead, we create updated clusters of

Circuit	*Gates*	*Lit flat*	*Lit part*	$Delay_{layout}$	$Avg\ Cong_{time}$	$Avg\ Cong_{layout}$	$Max\ Cong_{time}$	$Max\ Cong_{layout}$	CPU_{layout}
design1	138332	375302	382698	-0.01	24.3%	22.2%	122.1%	103.2%	0.28
design2	92582	244417	249060	-0.04	37.1%	32.2%	82.7%	78.1%	0.46
design3	43145	127749	132521	0.00	51.2%	48.2%	153.1%	127.3%	0.73
design4	101452	334822	342081	0.00	57.4%	52.2%	127.0%	121.8%	0.31
design5	97165	331383	336697	-0.02	48.2%	43.7%	121.1%	112.3%	0.29
design6	84292	320308	326646	0.00	39.2%	37.9%	101.7%	98.1%	0.32
design7	77215	221437	224651	0.00	47.1%	44.4%	158.5%	132.3%	0.35

Table 1: Optimization Results

bigger size, now using the actual layout locations of the objects in the netlist. We now have a technology-mapped pre-placed design and continue global placement using our quadratic placement algorithm. Finally, we uncluster the netlist and perform detailed placement and legalization.

5 Experiments and Results

The proposed integrated synthesis and placement flow has been implemented in C++ within the framework of the logic synthesis tool BooleDozer™ [13]. We employ the proposed layout aware decomposition and technology mapping algorithms in conjunction with partitioning for synthesis and clustering for placement. During decomposition, we use a combination of the exact branch and bound based algorithm for small number of inputs and our greedy algorithm for larger ones. Table 1 presents results of the layout driven optimization process for a number of IBM ASIC designs. *Lit flat* and *Lit part* show the amount of literals after logic minimization on the flat and synthesis partitioned design, respectively. On average, optimization of the partitioned design results in less than 2% more literals compared to the flat design, achieved due to the high degree of functional disjointness of the individual partitions. $Delay_{layout}$ represents the increase (or decrease) of the delay of the most critical path during layout driven optimization in comparison to the purely timing based decomposition and technology mapping. *Avg Cong* and *Max Cong* show the average and maximum horizontal and vertical congestion, for layout driven and purely timing based optimization. Due to the decreased amount of wire length and by avoiding local areas of high congestion, achieved by grouping only close objects, we can generally improve congestion compared to the timing based only optimization. In addition, by balancing timing and layout information, we can improve congestion while achieving timing closure. Figure 7 shows the congestion maps for circuit *design1* after synthesis and placement in the standard flow in comparison with our layout driven approach. Congestion across the horizontal and vertical cuts is shown in different colors ranging from blue (lighter) for the lowest to red (darker) for the highest congestion. Of particular importance is the maximum congestion which determines the routability of a design, which is consistently improved by avoiding local areas of high congestion.

Furthermore, we show results of the overall speedup achieved by our partitioning and clustering approach. CPU_{layout} denotes the runtime of the integrated synthesis flow in comparison to a completely flat approach. In general, further run time improvement is possible but will usually result in a lower quality of results.

a) b)

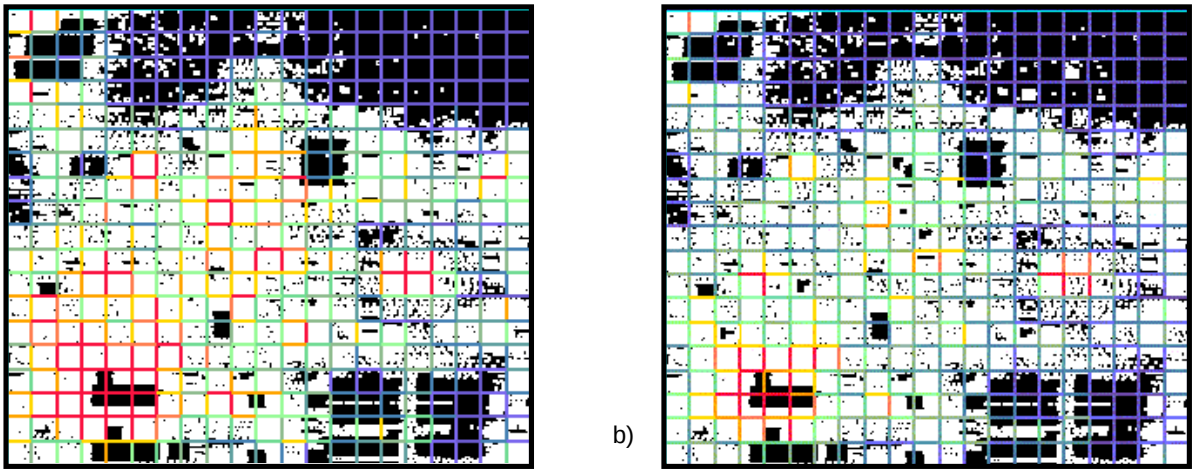

Figure 7: Congestion Maps for a) Standard Flow and b) Layout Driven Optimization

6 Conclusions and Future Work

The proposed placement and synthesis framework integrates the use of layout information for logic synthesis and the concept of co-existing synthesis partitioning and placement clustering to achieve better quality of results in a shorter time. We proposed congestion aware layout driven decomposition and technology mapping algorithms to effectively decrease overall wire length and improve congestion. In addition, we outlined the concept of co-existing synthesis partitions and placement cluster to reflect the different requirements of optimization in the logical and physical domains.

Future work will focus on the development of layout driven logic restructuring algorithms and the application of multi-level clustering during placement. In addition, the non-trivial task of parallelizing logic minimization of the synthesis partitions yields the potential for further speedup.

References

[1] T.F. Chan, J. Cong et. al. Multilevel Optimization for Large-Scale Circuit Placement. In *Proceedings of the International Conference on Computer Aided Design*, pp. 171-176, November 2000

[2] G. Karypis, R. Aggarwal, V. Kumar and S. Shekhar. Multilevel Hypergraph Partitioning: Application in VLSI Domain. In *Proceedings of the Design Automation Conference*, pp. 526-529, 1997

[3] G. Karypis and V. Kumar. Multilevel k-way Hypergraph Partitioning. In *VLSI Design*, Vol. 11, 3(2000), pp. 285-300

[4] D. M. Schuler and E. G. Ulrich. Clustering and Linear Placement. In *Proceedings of the Design Automation Conference*, 1972

[5] J. Cong and S. K. Lim. Edge Separability Based Circuit Clustering with Application to Circuit Partitioning. In *Proceedings of the Conference on Asia and South Pacific Design Automation, pp.429-434*, 2000

[6] H. Fujiwara and T. Shimono. On the Acceleration of Test Generation Algorithms. In *IEEE Transactions on Computers*, (32), pp. 1137-1144, December 1983

[7] F. Maamari and J. Rajski. A Reconvergent Fanout Analysis for Efficient Exact Fault Simulation of Combinatorial Circuits. In *Proceedings 18th Int. Symposium Fault Tolerant Computing*, June 1988

[8] S. Dey, F. Brglez and G. Kedem. Corolla Based Circuit Partitioning and Resynthesis. In *Proc. Design Automation Conference*, pp. 607-612, June 1990

[9] J. Lou, W. Chen and M. Pedram. Concurrent Logic Restructuring and Placement for Timing Closure. In *International Conference on Computer-Aided Design*, pp. 31-36, 1999

[10] M. Pedram and N. Bhat. Layout Driven Logic Restructuring/Decomposition. In *Proceedings of the Int. Conf. on Computer Aided Design*, pp.134-137, 1991

[11] W. Donath, P. Kudva, P. Villarrubia, L. Stok et. al. Transformational Placement and Synthesis. In *Proceedings of The Conference on Design, Automation and Test in Europe*, pp. 194-201, 2000

[12] G. Stenz et. al. Timing Driven in Interaction with Netlist Transformations. In *Proceedings of the International Symposium on Physical Design, 1997*

[13] L. Stok, D. Brand, D. Kung et. al. Booledozer: Logic synthesis for ASICs. In IBM Journal of Research and Development, 40(4), pp. 515-547, July 1996

[14] L. Stok, M. A. Iyer and A. J. Sullivan. Wavefront Technology Mapping. In *Proceedings of the Conference on Design, Automation and Test in Europe*, pp. 108-113, 1999