ADDRESSING THE TIMING CLOSURE PROBLEM BY INTEGRATING LOGIC OPTIMIZATION AND PLACEMENT

by

Wilsin Gosti, Sunil P. Khatri
and Alberto L. Sangiovanni-Vincentelli

Memorandum No. UCB/ERL M00/66

1 December 2000

ADDRESSING THE TIMING CLOSURE PROBLEM BY INTEGRATING LOGIC OPTIMIZATION AND PLACEMENT

by

Wilsin Gosti, Sunil P. Khatri
and Alberto L. Sangiovanni-Vincentelli

Memorandum No. UCB/ERL M00/66

1 December 2000

ELECTRONICS RESEARCH LABORATORY

College of Engineering
University of California, Berkeley
94720

Addressing the Timing Closure Problem by Integrating Logic Optimization and Placement *

Wilsin Gosti   Sunil P. Khatri+   Alberto L. Sangiovanni-Vincentelli
Dept. of EECS, University of California, Berkeley, CA 94720
+ Dept. of ECE, University of Colorado, Boulder, CO 80309

Abstract

As technology scaling enables the integration of many millions of devices on a single IC die, conventional design flows, which treat logic synthesis and physical design separately, exhibit an inability to achieve timing closure. Timing closure problems occur when timing estimates computed during logic synthesis do not match with timing estimates computed from the layout of the circuit. In such a situation, logic synthesis and layout synthesis are iterated until the estimates match. The number of such iterations is becoming larger as technology scales. Timing closure problems occur mainly due to the difficulty in accurately predicting interconnect delay during logic synthesis.

In this paper, we present an algorithm that integrates technology dependent logic synthesis and global placement to address the timing closure problem. We introduce technology decomposition and technology mapping algorithms that interleave their logic operations with incremental and global placement, in order to maintain a consistent placement while the algorithm is run. We show that by integrating logic synthesis and placement, we avoid the need to predict interconnect delay during logic synthesis. We demonstrate that our scheme significantly enhances the predictability of wire delays, thereby solving the timing closure problem. This is the main result of our paper. Our results also show that our algorithms result in a significant reduction in interconnect delay.

1 Introduction and Previous Work

In conventional logic synthesis, the literal count of a circuit is minimized. While the literal count is a good indication of the number of nets in a circuit, it cannot be used to estimate the wire-length of a net. For this purpose, *wire-load models* are typically used. A wire-load model is a function that estimates the wire-length of a net given the number of pins on the net. Since no placement information is available during traditional logic synthesis, this estimate is not very accurate. Typically, different wire-load models are used for different circuits depending upon the circuit size and the fabrication process targeted for the circuit. However, wire-load models underestimate the wire-lengths of many nets. This is because the length of a net is estimated using only the number of pins connected to that net. To demonstrate this, we took *industry2*, a large circuit from the 1990 MCNC Layout Synthesis Benchmark circuits, and ran GORDIAN [8] and DOMINO [2] on it. We use GORDIAN to do global placement and DOMINO to do detailed placement. Figure 1 shows a scatter plot of the actual length of all nets in *industry2* (shown by the "+" marks) super-imposed with the estimated length of these nets (shown as a dotted line) using the wire-load model of Table 2. The x-axis represents the number of pins connected to a net and the y-axis represents the length of the net in microns. As seen from this figure, a large number of net lengths are underestimated. This is especially true for nets connecting a few pins. As a result, conventional logic synthesis would underestimate the delay of these nets. If there are only a few such nets, then the likelihood of quick timing convergence is higher. However, if there are many such nets, then the number of iterations can be large or the iterations may not converge at all, which is often called the *Timing Closure* problem. For larger circuits, the number of nets underestimated by the wire-load model can only get larger, hence aggravating the timing closure problem.

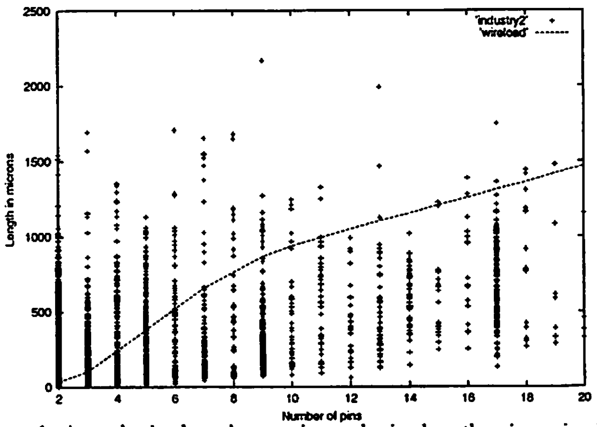

Figure 1: Actual wire length vs estimated wire length using wire-load model shown in Table 2.

Although there have been previous efforts to integrate logic synthesis and placement, only Shenoy *et al* [14] have specifically attempted to address the timing closure problem. In their work, they introduce a step *between* logic synthesis and global placement that iteratively improves the design by eliminating the maximum capacitance violations within the design. Our work is orthogonal to theirs since it *couples* logic synthesis and global placement. A number of approaches that integrate logic synthesis and placement have been proposed in the context of minimizing the parasitic delay of a circuit. Pedram and Bhat in [12] proposed a layout driven technology mapping scheme. While performing logic synthesis, a companion placement is maintained and the positions of nodes are used to estimate the wire-lengths of nets. The estimated length is then used as part of the cost function in the technology mapping step. The work of [12] is geared towards minimizing circuit delay, whereas the goal of our work is to solve the Timing Closure problem. Another significant difference is that in [12],

*This work was supported in part by SRC-DC-683.

different net models are used for global and incremental placement. In our work, we attempt to minimize the perturbation of the placement during logic operations by utilizing the same net models and algorithms for both global and incremental placement. This results in better delay and area characteristics for our algorithm. While [12] reported an area overhead while performing delay mapping, we obtain an 19% average area reduction instead. Pedram and Bhat in [11] proposed a layout driven technology *independent* optimization procedure (unlike our work, where we focus on technology *dependent* optimization). Again, a companion placement is maintained and the positions of Boolean nodes are used to drive kernel extraction and elimination algorithms. Lou, et al [10] presented an approach that integrates technology mapping and linear placement. This approach uses channel density as a cost function, and therefore only works when 2 routing layers are available. There have also been numerous approaches that restructure logic after placement like [17] and [9]. These are fundamentally different from our work since we *couple* technology mapping and global placement.

Figure 1 essentially shows that it is difficult if not impossible to estimate the length of a net during traditional logic synthesis. We therefore propose an approach which integrates logic synthesis and global placement so that the estimation of net lengths is not required at all. In particular, we look into the integration of technology mapping and global placement, using efficient placement models and algorithms. We believe that the key to the integration of logic synthesis and placement lies in the ability to perform *incremental* placement throughout the integrated optimization procedure such that at any point in the algorithm, the placement closely mimics the final placement. Using global and incremental placement algorithms that have the same net model and basic algorithms, we carefully interleave technology decomposition and technology mapping with placement such that the placement solution is perturbed minimally throughout the logic operations.

In this work, we show that the Timing Closure problem occurs due to the difficulty in estimating net lengths during logic synthesis (as an example, we demonstrate the ineffectiveness of the wire-load model, which is widely used to estimate wire-length during logic synthesis). The main contribution of this work is to successfully integrate a state-of-the-art global placement algorithm with SIS algorithms for minimizing area plus wire-length and delay cost functions. In this manner, we minimize Timing Closure problems. In addition, we show that on average, the circuit delay is reduced by using our technique. In case we perform delay mapping, we obtain an average area reduction of 19%.

The rest of the paper is organized as follows. In Section 2, we describe the technology and wire-load model used in this paper. In Section 3, we describe the incremental and global placement algorithms used. We describe in detail the integration of technology decomposition and technology mapping with placement algorithms in Section 4. In this section, we describe integrated algorithms with area plus wire-length and delay cost functions. We show our experimental results in Section 5. Finally, we describe future work and conclude in Section 6.

2 Technology and Net Models

2.1 Technology Parameters

For this paper, we use the $0.1\mu m$ "strawman" process technology developed by Khatri and Mehrotra [7]. Table 1 shows the parameters for this $0.1\mu m$ technology (for metal layers 1, 2, 3, and 4). Also listed in the table are the resistance per square and capacitance per micron (of a minimum-width wire). The capacitance value is the sum of the line-to-line (lateral) capacitance and the capacitance of a line to layers above and below it. These capacitances were obtained by using a 3-D parasitic extractor called Space3D [18].

Table 1: "Strawman" process parameters for $0.1\mu m$ technology.

Process (μm)		0.1
V_{DD} (V)		1.2
L_{eff} (nm)		50
t_{ox} (nm)		3
# Metal Layers		8
M1–2	H (μm)	0.26
	W (μm)	0.13
	space (μm)	0.13
	t_{ins} (μm)	0.32
	$r(\Omega/\square)$	0.085
	$c(fF/\mu m)$	0.07412
M3–4	H (μm)	1.0
	W (μm)	0.5
	space (μm)	0.5
	t_{ins} (μm)	0.90
	$r(\Omega/\square)$	0.0224
	$c(fF/\mu m)$	0.0543
Gate	$C_{eff}(fF)$	14
ε_r		2

2.2 Wire-Load Model

To account for interconnect during logic synthesis, wire-load models are conventionally used. Based on the number of pins connected to any net, a wire-load model returns the estimated wire-length of that net (net length). The wire-load model used in this paper is shown in Figure 2. The wire-length of an n-pin net is computed as m_n times the estimated wire-length of a 2-pin net.

Table 2: Wire-Load Model

n-pin	Multiplier (m_n)
2	1
3	3
4	7
5	11
6	15
7	19
8	22
9	25
10	27

For any net with more than 10 pins, the model follows a straight line with a slope of 1.6. We estimated the length of a 2-pin net statistically. We placed the four largest designs from the 1990 and 1992 MCNC Layout Synthesis benchmark circuits (industry2, industry3, avq.small, and avq.large) using GORDIAN and DOMINO. We assumed that all

2-pin nets are routed with their minimum length, i.e. the Manhattan distance between the two pins on the nets. We then compute the average length of all 2-pin nets over the four circuits and use it as the estimated length of a 2-pin net for the wire-load model of Table 2. This wire-length was determined to be 34.77μm.

Regardless of the specific wire-load model utilized, the basic problem with a logic synthesis methodology that depends on such statistical information to estimate wire delays is that it is prone to inaccuracies. Therefore there is a large probability that delay estimates extracted from the layout would be different from those generated during logic synthesis, using a wire-load model.

2.3 Net Model

Before the actual routing is performed, the topologies of nets are not known and need to be estimated in order to compute the delay of the circuit. In this paper, the topology of a net is estimated by a steiner tree. For efficiency reasons, the center of gravity of all the cells connected to the net is estimated to be the steiner point. An example is shown in Figure 2. In this figure, p is the steiner point. The length of this net is estimated to be the sum of all the net segments. With this topology model, the Elmore delay [4] is used to estimate the delay of the net. Each segment is modeled as a π-circuit, as shown in Figure 3.

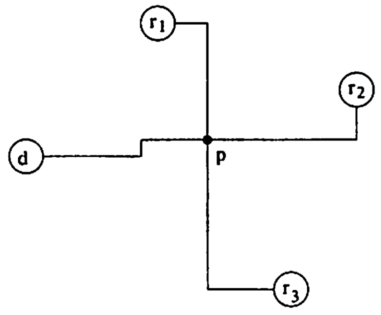

Figure 2: Topology model of a net.

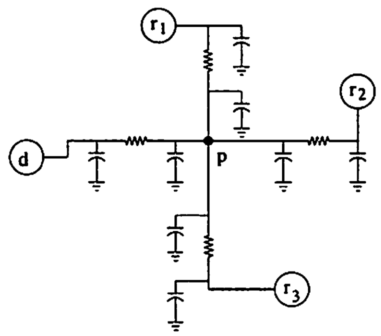

Figure 3: Delay model of a net.

3 Placement

For the placement of Boolean nodes in our scheme, we used the "Kraftwerk" global placement algorithm developed by Eisenmann [3]. In our approach, we integrate this algorithm into our logic synthesis tool, SIS [13]. Here we briefly describe how this algorithm works. In general, quadratic placement algorithms consist of two phases: solving the quadratic programming problem and spreading the cells apart to minimize overlaps [3] [16]. The former phase is similar across different quadratic placement algorithms. All nets are usually modeled as cliques to create a placement graph. The objective is to minimize the sum of the squares of the length of all edges in the placement graph. The formulation reduces to solving the linear equation $\mathbf{Ax} = \mathbf{b}$. The distinguishing part among various quadratic placement algorithms is usually the spreading phase. In [3], the spreading apart of nodes is achieved by introducing additional forces iteratively. In each iteration, every cell is subjected to an attracting force from every point in the space which is not occupied by a cell. With this main idea, the authors formulate and solve the additional forces mathematically. The result of adding these forces is a new right-hand-side vector. The net effect of this is that cells that are closest to an open space are likely to be pulled in to occupy that space. The important benefit of this approach over other quadratic placement algorithms is that it can be used in an incremental manner because only the $\mathbf{b}$ vector changes when additional forces are added. The algorithm can then continue spreading the cells apart until minimum overlap is achieved. We believe that the key requirement of integrating logic synthesis and placement is the ability to maintain a similar placement throughout logic optimization. The incremental nature of this quadratic programming based algorithm helps fulfill this requirement.

In practice, we cannot repeat the placement of all nodes in the Boolean network for every logic operation and cost function while our algorithm performs its computation. This would result in excessive run time. For cost computation, and when the Boolean network is minimally perturbed, we *incrementally* place the affected nodes. The requirement we place on the incremental placement is that the incremental placement results during logic operations and the final placement result are similar. To achieve this, the net model and algorithm used for the incremental placement need to be the same as those used in the global placement algorithm. Since we use a quadratic global placement tool for the final placement, we incrementally place nodes by formulating this problem as a quadratic programming problem as well. Because the net model used in the quadratic global placement tool is the *clique model*, we use the same clique model in our incremental placement algorithm. The clique model is illustrated in Figure 4. Let node n shown in Figure 4(a) be a new node, generated during logic synthesis optimizations. Let the positions of all its fanin and fanout nodes be known. We first model all fanin and fanout nets of n as cliques. The corresponding placement graph is shown in Figure 4(b). The quadratic programming problem is then formulated on this incremental placement graph. Since node n is the only node without a position, the problem can be solved in constant time.

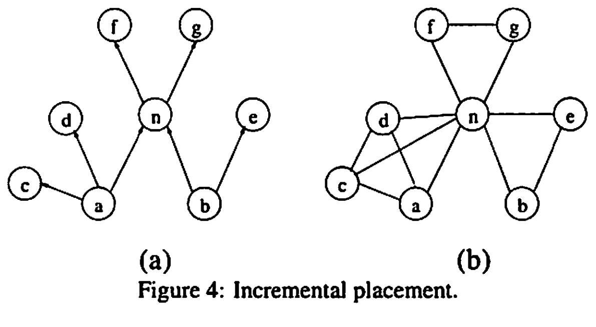

Figure 4: Incremental placement.

The idea of performing incremental placement with the same quadratic placement formulation and same net model as the global placement algorithm is a contribution of this work. Our results show that this results in a significant reduction in interconnect delay, while solving the timing closure problem as well.

4 Technology Dependent Optimization

Logic synthesis is typically divided into two optimization steps, technology independent optimization and technology dependent optimization [13]. In this paper, we address technology dependent optimizations. The technology dependent optimizations we cover in this work are: technology decomposition and technology mapping. Technology decomposition is the process of decomposing a Boolean network (representing the circuit to be implemented) into primitive gates, e.g. 2-input NOR gates and inverters. During technology mapping, the decomposed Boolean network (which consists of only primitive gates) is mapped into library gates.

4.1 Technology Decomposition

Our placement-aware technology decomposition algorithm decomposes a Boolean network using primitive gates, in a manner that minimizes wire-lengths in the decomposed network. The algorithm consists of the following steps:

1. We first invoke the global placement algorithm to find the positions of all nodes in the original Boolean network.

2. Next we decompose every Boolean node N of the original network into AND nodes (corresponding to each cube of the Boolean node N) and an OR node with all the AND nodes as its fanins. After all the nodes of the original network have been decomposed in this manner, we compute the positions of the new AND and OR nodes by invoking the global placement algorithm. Let the AND nodes be called $N_0, N_1, \cdots, N_{m-1}$, and let the OR node be called N_m. Here, the cardinality of the sum-of-products cover representing the logic function of N is m.

3. For each AND or OR node $n \in \{N_0, N_1, \cdots, N_m\}$ with fanins $FI = \{f_1, f_2, \ldots, f_k\}$, we decompose n into n' with fanins $NI = \left\{n_1, n_2, \ldots, n_{\lceil \frac{k}{2} \rceil}\right\}$, where $f_i \in FI$. After such a decomposition step, each node $n_j \in NI$ is a 2-input AND or OR node with a pair of nodes in FI as its fanins. We call this decomposition a *two-step decomposition.*

   The objective of the two-step decomposition process is to choose a pair of fanins for each n_j, so as to minimize the total wire-length of all the nets connected to the outputs of NI and FI, where the positions of nodes NI are computed utilizing the incremental placement algorithm. We call this problem the *fanin ordering* problem.

   Figure 5 illustrates this process. In this figure, all nodes are drawn to scale. Figure 5(a) shows a node N in the original Boolean network being decomposed. Figure 5(b) shows the decomposition of N into AND and OR nodes as described in step 2. The logic function computed by the node N is $f_1 f_2 f_3 f_4 f_5 f_6 f_7 f_8 + f_9 f_{10}$. Figure 5(c), shows node N_0 before two-step decomposition. For consistency in notation, n is also used to refer to N_0 in Figure 5(c). The result of two step decomposition is shown in Figure 5(d). The fanin ordering problem essentially aims to find a two-step decomposition of the nodes in Figure 5(c) such that the sum of wire-lengths of all wires in the resulting decomposition (Figure 5(d)) is minimized.

   At the end of this step, not all nodes are 2-input nodes. For example, node n' in Figure 5(d) has 4 fanins.

4. After all AND and OR nodes in the network have been decomposed using the two-step decomposition method, we run the global placement algorithm to update all node positions.

5. We then iterate steps 3 and 4 over all Boolean nodes of the network until all nodes have at most two fanins.

6. Finally, we run global placement on the resulting network of primitive gates. The reason for running the global placement at this stage is to legalize the resulting incremental placement. Since the core algorithm and net model of both the incremental algorithm and the global placement algorithm are the same, the resulting placement is minimally perturbed.

In addition to the invocations of the global placement algorithm described above, we additionally invoke global placement at most β times (where β is a user defined variable) during the technology decomposition algorithm. This is to ensure that our incremental placement runs utilize accurate node placement information at all times. We experimented with several values of β, and found that $\beta = 10$ resulted a good trade-off between run-time and circuit optimality.

Since many nodes are created during technology decomposition, all nodes are treated as points during global placement. This is handled by assigning a fixed size cell for every node in the network. This size is scaled according to the number of nodes in the network such that the total area matches the available placement area. This results in minimum overlap throughout technology decomposition.

Theorem 4.1 *The fanin ordering decision problem is NP-complete.*

Proof: In the decision problem, we ask the question if a decomposition whose total wire length is less than a constant B exists. The problem is clearly in P because we can compute the total wire length given a decomposition. Let n_{ij} be the new node obtained by pairing $f_i \in FI$ and $f_j \in FI$. The position of n_{ij} can be computed using incremental placement as described above. Let $d(f_i, f_j)$ be the cost of pairing f_i and f_j. The cost $d(f_i, f_j)$ is the sum of the length of nets f_i, f_j, and n_{ij}. We perform reduction from the *clustering* problem [5]. Let the finite set X of the clustering problem be the set of nodes FI. Let the distance between any pair (x_i, x_j) of X be $d(f_i, f_j)$ as described above. Then it is easy to see that there exists a partition of X into $\lceil \frac{k}{2} \rceil$ disjoint sets such that the total distance is $\leq B$ iff there is a decomposition whose total wire length is $\leq B$. ■

Since the fanin ordering problem is a hard problem, we utilize heuristics to solve it. The two heuristics that we use are *angle* ordering and *furthest-pair* ordering. In both heuristics, we look for a linear order $FI_s = (f_{s_1}, f_{s_2}, \cdots, f_{s_k})$ of FI. Then nodes NI are created by pairing nodes in FI_s in this linear order.

In *angle ordering*, we traverse the fanins of node n in a counter clockwise direction to form a circular order. The two consecutive fanins in this traversal that are furthest apart in terms of linear distance form the end points of the final linear order. The remaining nodes are ordered in the counter clockwise manner. For the example shown in

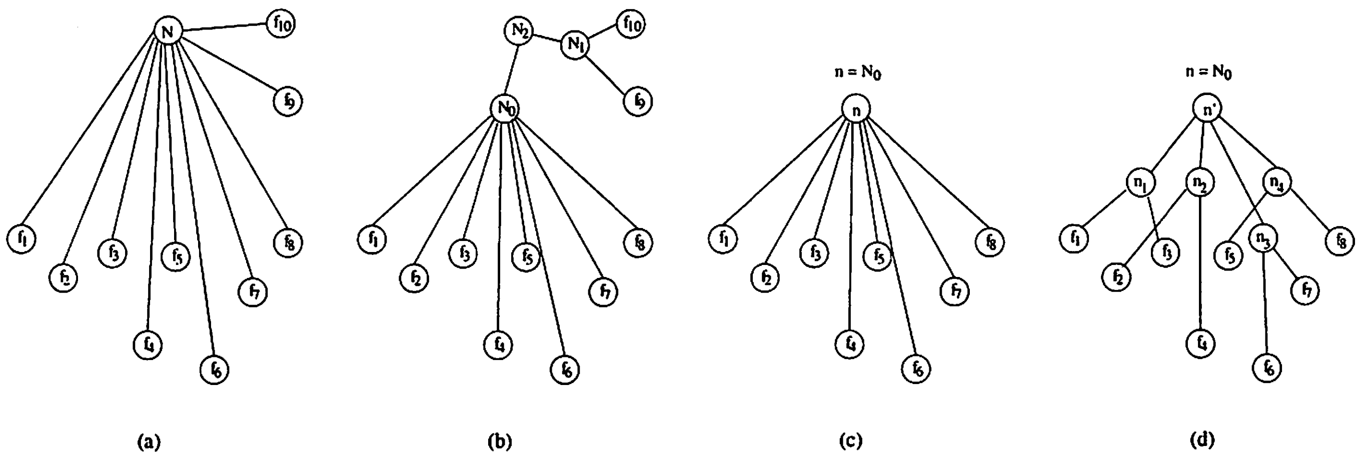

Figure 5: Technology Decomposition and Fanin Ordering Problem.

Figure 5(c), the counter clockwise traversal gives the following circular order: $(f_1,f_2,f_3,f_4,f_5,f_6,f_7,f_8,f_1,...)$. The two consecutive fanins that are furthest apart in terms of linear distance are f_1 and f_8. Hence these two nodes form the two end points of the linear order, and this linear order is therefore $(f_1,f_2,f_3,f_4,f_5,f_6,f_7,f_8)$. The angle decomposition of the example of Figure 5(c) is shown in Figure 6.

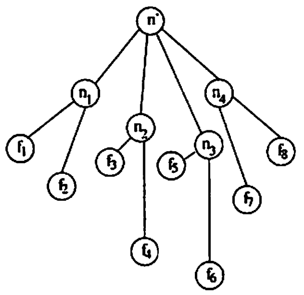

Figure 6: Angle Ordering Solution.

In *furthest pair* ordering, we iteratively pair fanin nodes (i.e. nodes in *FI*) until there are no more nodes to pair. In each iteration, we first find the fanin $f_f \in FI$ that is furthest away from node n in terms of linear distance. We then pair node f_f with the fanin node $f_g \in FI$ that is closest to f_f in terms of linear distance. Nodes f_f and f_g form the inputs to a new 2-input node in NI. The result of furthest pair ordering for the example given in Figure 5(c) is shown in Figure 7.

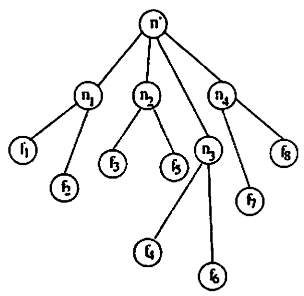

Figure 7: Furthest Pair Solution.

For each node, we perform technology decomposition using both the angle order and the furthest pair order and compute the cost of each order in terms of total net lengths. The cost of an order is the sum of the length of the fanin nets and the new nets (i.e. the total length of the nets in Figures 7 and 6).

4.2 Technology Mapping

After technology decomposition, the original Boolean network is transformed into a network consisting exclusively of primitive gates. This network is called the *subject graph*. The gates in the library are also decomposed using the same primitive gates, and each such decomposed gate is called a *pattern graph*. Technology mapping is the process of covering the subject graph with the pattern graphs while minimizing an objective function. For area minimization, the objective function is the total area of the mapped circuit. For delay minimization, the objective function is the total delay of the mapped circuit. If the subject graph is a tree, technology mapping with the objective function of area minimization can be solved optimally using dynamic programming [6].

We describe our area and delay minimization algorithms below:

4.2.1 Area and Wire-Length Minimization

In our approach, we decompose the subject graph into trees and use dynamic programming to solve it. The dynamic programming based algorithm consists of two steps: the forward propagation step to compute the cost of a best match and to store it on the node, and the backward tracing step to construct the match.

The forward propagation step traverses the trees in topological order from primary inputs to primary outputs such that optimum matches for all fanins of a node n are found before a match for n is found. Let a match m at a node n use a gate g from the library. Also let n_m be the new node for match m. Let $FI = \{f_1, f_2, \ldots, f_k\}$ be the fanins of n_m. We recursively define the *fanin wire cost* $w_l(n_m)$ of match m as the total length of the fanin nets of n_m plus the fanin wire cost of all its fanins. Formally, $w_l(n_m) = \sum_{f_i \in FI} w_l(n_{f_i}) + l(n_{f_i})$, where n_{f_i} is the node of the best match at f_i and $l(n_{f_i})$ is the length of its net. Note that net n_{f_i} is a two-terminal net, since the network being mapped consists of primitive gates which are 2-input gates, and we operate on tree decompositions of this network. Essentially, $w_l(n_m)$ is the total wire length of all nets in the mapped circuit rooted at node n_m.

Similarly, we recursively define the *area cost* $a(m)$ of match m as the area of g plus the total area of all its fanin matches. The total cost $c(m)$ of match m is then the weighted sum of the area cost of m and the sum of the fanin wire cost of m plus the length of net n, or $c(m) = a(m) + \alpha(w_f(m) + l(n))$, where α is a user-defined weighting variable.

Figure 8 illustrates these definitions. The fanin wire cost of match m is the sum of the wire lengths of nets w_1, w_2, w_3, and the fanin wire costs of the matches of f_1, f_2, and f_3. The area cost of match m is the sum of the area of g and the area costs of the matches for f_1, f_2, f_3.

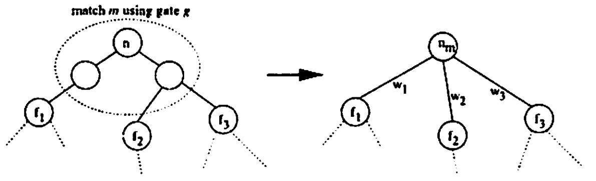

Figure 8: Definition of Cost Elements.

In order to compute the cost of a match, the position of the new node corresponding to the match needs to be computed (i.e. the node which shown in the dotted line in Figure 8). As in technology decomposition, we could re-run global placement on all nodes in the design (including the new node). However, this is too time consuming and so we use the incremental placement algorithm described in Section 3 to estimate the position of the new node.

Even though the incremental placement algorithm uses the same net model and the same quadratic placement based formulation as the global placement algorithm, after a certain number of nodes have been matched, the global placement algorithm is run on the entire circuit. Just as in technology decomposition, the user defined parameter β is used here. During the whole technology mapping algorithm, the global placement algorithm is called at most β times. All other executions of the placement algorithm during technology mapping are in incremental mode.

During technology mapping, not all nodes are mapped in general when calling global placement. The unmapped nodes are primitive nodes: NAND/NOR and inverters. Before calling global placement, NAND/NOR and inverter nodes are temporarily mapped into NAND/NOR and inverter gates in the library. The temporarily mapped network is then placed. As in technology decomposition, all cells are scaled to match available placement area to minimize overlap.

4.2.2 Delay Minimization

For delay minimization, we integrate the delay mapping algorithm developed by K. J. Singh [15] with global placement. Like area minimization, this algorithm decomposes the subject graph into trees and use dynamic programming to find a good mapping.

Unlike the conventional area minimization algorithm, the dynamic programming approach is not guaranteed to produce an optimum solution. The reason is that the arrival time of a cell is a function of not only the arrival time its fanin cells, but also the load due to its fanout cells. Hence, the optimality condition of a dynamic programming algorithm cannot be met. A good solution can be approximated by estimating the load due to its fanout cells in the forward propagation step. Singh has improved this algorithm by intelligently storing a set of matches as opposed to a single match for each node in the forward propagation step. The set of matches is stored in the form of a piece wise linear function. During the backward tracing step, the load is known and a match can be found by evaluating the piece wise linear function stored in the node. For detail of this algorithm, we refer the readers to his thesis [15].

In our work, we interleave Singh's approach with the incremental/global placement algorithm that has been described earlier.

5 Experimental Results

The library we use is the MSU standard cell library [13]. We modified the library based on resistances and capacitances obtained from SPICE characterizations (using the $0.1\mu m$ process technology described in Section 2). We use DOMINO [2] as our detail placement tool. Routing is done using the WARP [1] router from CADENCE. Although our 0.1μ technology has as many as 8 layers of metal, we only use 4 layers for signal routing purposes. We assume all other layers are used for routing of power and global signals. In all our experiments, we use the interconnect delay model shown in Figure 3.

The experimental setup is as follows. For area and wire-length minimization, we ran area optimization on our benchmark circuits using *script.rugged* in SIS [13]. Then we performed technology decomposition and technology mapping using SIS and the proposed scheme. Since we decompose our subject graph into trees before technology mapping, we compare our results with the corresponding algorithm in SIS. The parameters α and β used in the experiments are 0.1 and 10 respectively. For delay minimization, we ran delay optimization on our circuits using *script.delay* in SIS [15] and using our proposed scheme. The SIS algorithms we compared our work against utilized the same options as were utilized by our algorithm.

Since we address the timing closure problem, we first show (in Table 3) the delay of the circuits using the wire-load model of Section 2 and the actual delay of these circuits obtained after detail routing. This experiment illustrates the severity of the timing closure problem using traditional logic synthesis. The total delay is the delay of the critical path, and interconnect delay is the total delay including wire delay minus the total delay excluding wire delay. In this table, a negative number means that the actual number is smaller than the estimated one. As seen from this table, the wire-load model (the columns labeled "Wire") over-estimates the actual delay in some cases and under-estimates it in others. For the example *dalu*, the error in the estimated interconnect delay is as high as 57% and the error in the estimated total delay is 11%. The average error in the estimated interconnect delay is 29.0% and the average error in the estimated total delay is 4%. These averages are computed using the absolute values of the errors of all circuits. For the proposed approach (the columns labeled "PILS"), except for *dalu*, the estimated interconnect delay is small and the estimated total delay is within 1% of the actual delay. This tables shows the effectiveness of our integrated approach.

Table 4 reports the delay comparison between technology mapping minimizing area using SIS versus our approach (the column labeled "PILS"). In addition to the advantage of not having to estimate net lengths (and thereby solving the Timing Closure problem as shown in Table 3), our scheme exhibits a significant reduction in interconnect delay as seen from Table 4. For the example *dalu*, this translates into a 29% reduction in total delay. Using our scheme, the average reduction in interconnect delay is 17.7%, and the average reduction in total delay is 5.2%.

The improvement in interconnect delay of our method is accompanied by a small penalty in active area, as shown in Table 5. The average penalty in active area is 3.5%.

Table 3: Estimated delay vs actual delay using wire-load model and our approach.

Name	Interconnect Delay		Total Delay	
	Wire	PILS	Wire	PILS
C1908	−4%	0%	0%	0%
C2670	8%	−2%	1%	0%
C3540	26%	2%	3%	0%
C432	41%	−6%	5%	−1%
C499	−3%	−8%	0%	−1%
C880	−52%	−4%	−3%	0%
b9	−29%	0%	−2%	0%
dalu	57%	−34%	11%	−8%
k2	44%	0%	7%	0%
Ave	29%	6%	4%	1%

Table 4: Area and wire-length minimization.

Name	Interconnect Delay (ps)			Total Delay (ps)		
	SIS	PILS	Change	SIS	PILS	Change
C1908	107	88	−21%	1353	1303	−4%
C2670	158	134	−18%	1451	1393	−4%
C3540	309	199	−56%	2209	1938	−12%
C432	160	171	7%	1563	1613	3%
C499	66	64	−4%	868	948	9%
C6288	319	347	8%	4981	4646	−7%
C880	118	99	−19%	1732	1564	−10%
b9	33	27	−23%	312	306	−2%
dalu	547	354	−55%	2486	1764	−29%
k2	183	191	5%	1154	1182	2%
Ave	–	–	−17.7%	–	–	−5.2%

Table 5: Area of area and wire-length minimization (in μ^2)

Name	SIS	PILS	Change
C1908	59.33	61.40	3%
C2670	78.62	83.58	6%
C3540	148.44	157.42	6%
C432	23.33	23.73	2%
C499	58.18	56.39	−3%
C6288	368.58	359.25	−3%
C880	47.58	50.52	6%
b9	15.26	14.98	−2%
dalu	107.83	118.48	10%
k2	132.60	138.70	5%
rot	81.56	87.72	8%
Ave	–	–	3.5%

Table 6 shows the delay comparison between technology mapping minimizing delay using SIS versus our approach. Again, we achieve a significant reduction in interconnect delay and total delay. Table 7 shows the area comparison of SIS and our delay minimization scheme. As seen from this table, we achieve significant reduction in area in addition to the delay reduction seen from Table 6. We save an average of 19% in area. So, in addition to minimizing Timing Closure problems, our algorithm is able to reduce total circuit delay as well as total circuit area.

Table 6: Delay minimization.

Name	Interconnect Delay (ps)			Total Delay (ps)		
	SIS	PILS	Change	SIS	PILS	Change
C1908	183	164	−11%	1419	1391	−2%
C2670	258	230	−11%	1937	1811	−6%
C3540	452	385	−15%	2603	2243	−14%
C432	165	132	−20%	1607	1408	−12%
C499	61	66	8%	830	806	−3%
C6288	280	299	7%	3720	3559	−4%
C880	108	90	−16%	1194	981	−18%
b9	24	17	−30%	361	299	−17%
dalu	618	524	−15%	2589	2277	−12%
k2	389	304	−22%	1283	1211	−6%
rot	97	104	8%	806	942	17%
Ave	–	–	−10.6%	–	–	−7.1%

Table 7: Area of delay minimization. (in μ^2)

Name	SIS	PILS	Change
C1908	139.16	113.30	−19%
C2670	188.87	133.17	−29%
C3540	281.55	223.83	−20%
C432	38.94	36.00	−8%
C499	85.31	66.76	−22%
C6288	707.67	607.10	−14%
C880	88.70	72.40	−18%
b9	22.92	19.30	−16%
dalu	254.76	197.28	−23%
k2	353.89	273.25	−23%
rot	144.98	120.38	−17%
Ave	–	–	−19%

6 Conclusions and Future Work

In this paper, we have presented an approach that addresses the timing closure problem in IC design. Our approach integrates the technology mapping step of logic synthesis with placement. We believe that success in integrating logic synthesis and placement is dependent on the ability to maintain a consistent placement during logic synthesis which closely approximates the final placement. We used incremental and global placement algorithms to achieve this goal.

We have introduced technology decomposition and technology mapping algorithms using this integrated flow. We attempt to minimize a weighted function of area and wire length while interleaving incremental and global placement with logic operations. We also implemented a delay minimization algorithm.

The benefits of our approach are:

• The main result of our paper is the demonstration of a significant reduction in Timing Closure problems. Timing closure results in traditional logic synthesis underestimating interconnect delay by 29% on average. Our scheme reduces this error to 6%.

• Both incremental and global placement algorithms utilize the same core placement algorithm, and the same net model. This helps maintain a consistent placement during logic operations.

• Additionally, our scheme results in average reductions of about 17.7% in interconnect delay, and about 5.2% in total circuit delay for area and wire-length minimization. This is accompanied by an area penalty of 3.5%.

• For delay minimization algorithm, we achieve an average reduction of about 10.6% in interconnect delay, and about 7.1% in total circuit delay. In addition to this, we gain about 19% in area.

Based on the promising results of this paper, we intend to extend the current approach to include technology independent optimization as well.

References

[1] Cadence Design Systems, Inc., 555 River Oaks Parkway, San Jose, CA 95134, USA. *Envisia Silicon Ensemble Place-and-route Reference*, Nov 1999.

[2] K. Doll, F.M. Johannes, and G. Sigl. Domino: deterministic placement improvement with hill-climbing capabilities. *Proceedings of the IFIP International Conference on VLSI*, pages 91–100, August 1991.

[3] H. Eisenmann and F.M. Johannes. Generic global placement and floorplanning. In *DAC*, pages 269–274, June 1998.

[4] W. C. Elmore. The transient analysis of damped linear networks with particular regard to wideband amplifiers. *J. Applied Physics*, (19):55–63, 1948.

[5] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-completeness*. W. H. Freeman and Company, 1979.

[6] K. Keutzer. DAGON: Technology binding and local optimization by dac matching. In *DAC*, pages 228–234, 1990.

[7] S Khatri, A Mehrotra, R Brayton, A Sangiovanni-Vincentelli, and R Otten. A novel VLSI layout fabric for deep sub-micron applications. In *Proceedings of the Design Automation Conference*, New Orleans, June 1999.

[8] J.M. Kleinhans, G. Sigl, F.M. Johannes, and K.J. Antreich. GORDIAN: VLSI placement by quadratic programming and slicing optimization. *IEEE Trans on CAD*, 10(3):356–365, March 1991.

[9] J. Lou, W. Chen, and M. Pedram. Concurrent logic restructuring and placement for timing closure. In *ICCAD*, pages 31–35, November 1999.

[10] J. Lou, A. H. Salek, and M. Pedram. An exact solution to simultaneous technology mapping and linear placement problem. In *ICCAD*, pages 671–675, November 1997.

[11] M. Pedram and N. Bhat. Layout driven logic restructuring/decomposition. In *ICCAD*, pages 134–137, 1991.

[12] M. Pedram and N. Bhat. Layout driven technology mapping. In *Proc. of the* 28^{th} *DAC*, pages 99–105, 1991.

[13] E. M. Sentovich, K. J. Singh, L. Lavagno, C. Moon, R. Murgai, A. Saldanha, H. Savoj, P. R. Stephan, R. K. Brayton, and A. L. Sangiovanni-Vincentelli. SIS: A System for Sequential Circuit Synthesis. Technical Report UCB/ERL M92/41, Electronics Research Lab, Univ. of California, Berkeley, CA 94720, May 1992.

[14] N. Shenoy, M. Iyer, R. Damiano, K. Harer, H. Ma, and P. Thilking. A robust solution to the timing convergence problem in high-performance design. In *The Proceedings of the International Conference on Computer Design*, pages 250–257, October 1999.

[15] Kanwar Jit Singh. *Performance Optimization of Digital Circuits*. PhD thesis, University of California, Berkeley, December 1992. Tech. Report No. UCB/ERL M92/149.

[16] A. Srinivasan, K. Chaudhary, and E. S. Kuh. Ritual: a performance driven placement algorithm. *IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing*, 39(11):825–840, November 1992.

[17] G. Stenz, B. M. Riess, B. Rohfleisch, and F. M. Johannes. Timing Driven Placement in Interaction with Netlist Transformations. In *ISPD 97*, Napa Valley, CA, 1997.

[18] A. J. van Genderen and N. P. van der Meijs. Space user's manual, space tutorial, space 3d capacitance extraction user's manual. Technical report, Delft University of Technology, Dept of EE, Delft, The Netherlands, 1995.