Addressing the Timing Closure Problem by Integrating Logic Optimization and Placement

Wilsin Gosti
Dept. of EECS
University of California
Berkeley, CA 94720

Sunil P. Khatri
Dept. of ECE
University of Colorado
Boulder, CO 80309

Alberto L. Sangiovanni-Vincentelli
Dept. of EECS
University of California
Berkeley, CA 94720

Abstract

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 logic synthesis and global placement to address the timing closure problem. We introduce technology independent algorithms as well as technology dependent algorithms. Our technology independent algorithms are based on the notion of "wire-planning". All these algorithms interleave their logic operations with local and incremental/full 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 total circuit delay. In addition, our technology independent algorithms result in a significant circuit area reduction.

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 [9] and DOMINO [2] on it. We use GORDIAN for global placement and DOMINO for 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). 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.

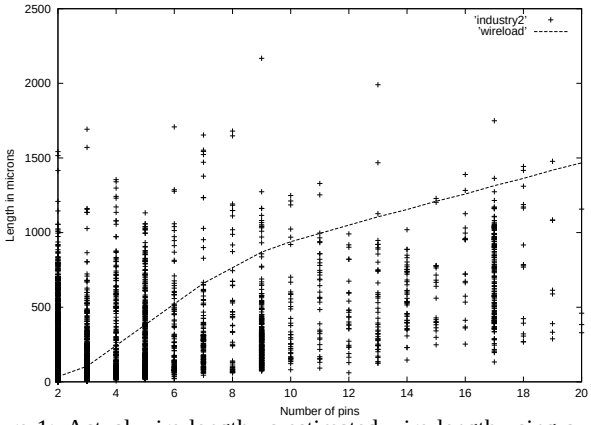

Figure 1: Actual wire length vs estimated wire length using a wire-load model.

There have been previous efforts to integrate logic synthesis and placement. Shenoy *et al* [15] 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 we *couple* logic synthesis and global placement. A number of approaches that integrate logic synthesis and placement have been proposed in the context of minimizing the interconnect delay of a circuit. Pedram and Bhat in [13] 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. In [12], Pedram and Bhat also proposed a layout driven technology independent optimization procedure. Again, a companion placement is maintained and the positions of Boolean nodes are used to drive kernel extraction and elimination algorithms. Our work is different from their work in several regards. In [13], 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 in-

cremental placement. This results in better delay and area characteristics for our algorithm. We also employ different technology dependent and independent algorithms in arriving at our results. For example, our technology independent duplication and kernel extraction algorithms are heuristics based on the wire-planning approach of [6]. Additionally, the area and delay results we obtain are better. Lou, et al [11] 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 [10]. 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 both technology independent and technology dependent optimization with 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 our optimization algorithms with placement such that the placement solution is perturbed minimally throughout the logic operations.

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 optimization, we obtain an average area reduction of 18%.

The rest of the paper is organized as follows. In Section 2, the technology and net model used in this paper are described. Section 3 describes the incremental and global placement algorithms used. We describe the integration of technology independent optimization with placement in Section 4. Section 5 covers the integration of technology dependent optimization with placement. We show our experimental results in Section 6. Finally, we conclude in Section 7.

2 Technology and Net Models

2.1 Technology Parameters

For this paper, we use the 0.1μm "strawman" process technology developed by Khatri and Mehrotra [8]. Table 1 shows the parameters for this 0.1μm 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 [19].

2.2 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. The delay from the driver to any point on the net is computed using Elmore delay [4].

Table 1: "Strawman" process parameters for 0.1μm 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
	$r(\Omega/\Box)$	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
	$r(\Omega/\Box)$	0.0224
	$c(fF/\mu m)$	0.0543
Gate	$C_{eff}(fF)$	14
ε_r		2

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 [14]. 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 **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 *locally* place the affected nodes. We require that the local placement results during logic operations and the final placement result are similar. To achieve this, the net model and algorithm used for local placement are the same as those used in the global placement algorithm. Since we use a quadratic global placement tool for the final placement, we locally place nodes by formulating this problem as a quadratic programming problem as well. Because the net model used in the quadratic global placement step is the *clique model*, we use the same clique model in our local placement algorithm. The clique model is illustrated in Figure 2. Let node *n* shown in Figure 2(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 2(b). The quadratic programming problem is then formulated on this local placement graph. Since node *n* is the only node without a position, the problem can be solved in *constant* time.

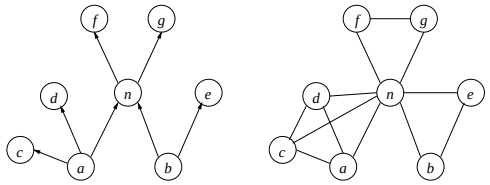

Figure 2: (a) A new node *n* and its neighbors. (b) The placement graph.

The idea of performing local 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.

In summary, we utilize three different placement algorithms. The first is the "Kraftwerk" global placement algorithm running in full mode, i.e. without a given initial placement. The second is the "Kraftwerk" global placement algorithm running in incremental mode, i.e. with a given initial placement and the placement is minimally perturbed. The third is the local placement algorithm where the position of a cell is determined by the position of all cells connected to its input and output nets as described in Figure 2.

4 Technology Independent Optimization

There are a number of logic operations commonly employed in the technology independent step, like kernel extraction, re-substitution, and simplification of Boolean nodes. Kernel extraction extracts common kernels from Boolean nodes and new nodes are created for the kernels. Re-substitution re-expresses a node in terms of other nodes in the Boolean network. Node simplification uses satisfiability and observability don't cares to simplify each Boolean node separately. Among these logic operations, kernel extraction changes the structure of the network extensively, allowing significant opportunities to improve the quality of the results. For this reason, we focus our attention on integrating kernel extraction and placement. In addition, we introduce a position based duplication algorithm.

In conventional synthesis, the cost function used in technology independent optimization is the literal count of the network. The kernel extraction algorithm iterates through the Boolean network. In each iteration, it traverses the network to find a kernel that reduces the literal count maximally and extracts it as a new node in the circuit. The Boolean network is then re-expressed using the new kernel. The iteration stops when the literal count of the network cannot be reduced further.

In this paper, the algorithm is similar but the cost function is modified to include an increase in literal count if a node is duplicated. The heuristic to duplicate a node is based on the wire-planning approach presented in [6]. This type of kernel extraction is referred to as *wire-planned* kernel extraction.

4.1 Value of a Kernel

The *value* of a kernel is the number of literals in the Boolean network that can be decreased if the kernel is extracted. The value $value(k)$ of a kernel k can be computed easily. First, express the Boolean function at each of the fanout nodes in factored form. Let n_k be the number of times kernel k appears in the factored form of all its fanouts. Also, let l_k be the number of literals in k. The value of k is

$$value(k) = n_k(l_k - 1) - l_k$$

because for each occurrence of k in the fanout, $(l_k - 1)$ literals are reduced and l_k literals are needed to represent kernel k as a new Boolean node in the network.

4.2 Wire-planned Kernel Extraction

In the wire-planning approach [6], paths from all primary inputs to all primary outputs are required to be monotonic[1]. The basic idea of the heuristic used here is to relax the monotonic constraint on a path by applying it locally.

4.2.1 Weight of a Kernel

For a node k with placement constraint $\{i_1, i_2, \ldots, i_m\}_{o_1,o_2,\ldots,o_l}$ (i.e. the transitive fanins of k include primary inputs $i_1, i_2, \ldots, i_m$, and the transitive fanouts of k include primary outputs $o_1, o_2, \ldots, o_l$), the *input box* of k is defined as the smallest rectangle that encloses the primary inputs of k and the *output box* of k is the smallest rectangle that encloses the primary outputs of k. With these definitions, the conditions under which a node is legal[2] in the wire-planning approach can be restated by the following lemma.

Lemma 1 *For a legal node k with placement constraint $\{i_1, i_2, \ldots, i_m\}_{o_1,o_2,\ldots,o_n}, m > 1, n > 1$, its legal region intersects with its output box only at its closest point, i.e. the point closest to any output. Similarly, its legal region intersects with its input box only at its furthest point, i.e. the point furthest from any output. (The closest and furthest points are unique for a legal node [6]).*

Proof: By the definition of *region* and the *intersection rule* in [6], the lemma follows. ■

The *primary output angle* of a node k is defined to be the angle subtended by k and all its primary outputs. Figure 3(a) illustrates this definition. By Lemma 1, the primary output angle of a legal node k is at most 90°, which is when k is placed at the closest point of its legal region. Similar to the definition of primary output angle, the *fanout angle* of a node k (denoted by α_k) is defined to be the minimum angle subtended by node k and any pair of its immediate fanouts, o_i and o_j,

[1] A path from a primary input *pi* to a primary output *po* is monotonic if the length of the path is equal to the Manhattan distance from *pi* to *po*.

[2] A node is *legal* if there exists a position such that if the node is placed in that position, all paths from any primary input to any primary output going through it remain monotonic.

such that every other fanout is visited in an angular traversal from o_i to o_j along the angle α_k.

Figure 3(b) illustrates this definition. In this example, an angular traversal along the angle α, subtended by o_1, k, and o_4, allows us to visit all other nodes in the fanout of the node k. In this example, α is the smallest angle such that every other fanout can be visited by an angular traversal around it. Therefore α is the fanout angle of node k.

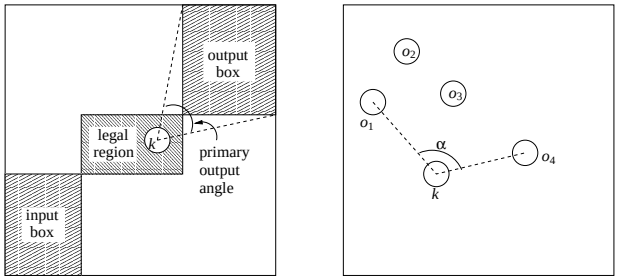

Figure 3: (a) Primary output angle of k. (b) Fanout angle of k.

The idea behind the wire-planned kernel extraction is to restrict the fanout angle α_k of a kernel k to be at most 90°. If α_k is larger than 90°, then the weight of the kernel is defined to be the value of the kernel minus the number of literals needed to duplicate the kernel in order to maintain a maximum fanout angle of 90° for each kernel and its duplicates. Hence, the weight $w(k)$ of a kernel k is

$$w(k) = value(k) - \left\lfloor \frac{\alpha_k}{90^\circ} \right\rfloor \times l_k$$

where l_k is the number of literals of k.

4.3 Wire-planned Duplication

In the wire-planned kernel extraction described above, the weights of kernels are used as the objective function of the algorithm. However kernels are not duplicated during kernel extraction. Boolean nodes can be very large after kernel extraction. Duplicating nodes at this point can increase the literal count considerably. The duplication process is therefore performed as a separate operation, which is called a *wire-planned duplication* operation.

The wire-planned duplication algorithm traverses the Boolean network and duplicates nodes whose fanout angles are larger than 90° Eventually, the fanout angles of the nodes and their duplicates are smaller than or equal to 90°. The algorithm is guaranteed to terminate because in the worst case each node only has a single fanout.

Let l be the number of fanouts of k and let o_1 and o_l be the two end points found when computing the fanout angle α_k of k. If $\alpha_k > 90^\circ$, then a circular order of the fanouts is computed, denoted as $FO^s(k) = (o_1, o_2, \ldots, o_l)$. The duplication algorithm partitions the set of all fanouts into $\left\lceil \frac{\alpha_k}{90^\circ} \right\rceil$ sets. The partitions are created by traversing the fanouts in the order of the sorted list $FO^s(k)$. A partition starts from o_1, and nodes are added until a fanout is reached that would result in a fanout angle larger than 90°. A new partition is then started and the procedure is repeated.

4.4 Interaction with Placement

The global placement is invoked several times during kernel extraction. Since the number of iterations is not known before the kernel extraction algorithm terminates, the number of global placement invocations cannot be determined in advance. Instead, a parameter γ is introduced. The incremental global placement algorithm is invoked after every γ iterations.

As mentioned above, the kernel extraction algorithm is an iterative algorithm. In each iteration, a single kernel is selected from all generated kernels. The costs of all kernels need to be computed, which means that the location of all kernels need to be estimated. For this purpose, the local placement algorithm described above is used.

During technology independent optimization, the circuit structure is typically very different from the final circuit generated after technology dependent optimization. It is therefore not necessary to use interconnect delay as one of the criteria in determining the best kernel to extract. However, the relative positions of Boolean nodes along input/output paths during kernel extraction will likely remain the same as the placement of the final circuit. For this reason, the cost function of interconnect length is preferred to interconnect delay in computing the weight of a kernel. As a result, the semi-perimeter estimate is used to compute the length of a net (instead of the more computationally expensive Steiner tree estimate).

5 Technology Dependent Optimization

The technology dependent optimizations typically consists of two steps: 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.

5.1 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 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 full 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.

2. 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 local placement algorithm. We call this problem the *fanin ordering* problem.

   Figure 4 illustrates this process. In this figure, all nodes are drawn to scale. Figure 4(a) shows a node N in the original Boolean network being decomposed. Figure 4(b) shows

the decomposition of N into AND and OR nodes as described in step 1. 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 4(c), shows node N_0 before two-step decomposition. For consistency in notation, n is also used to refer to N_0 in Figure 4(c). The result of two step decomposition is shown in Figure 4(d). The fanin ordering problem essentially aims to find a two-step decomposition of the nodes in Figure 4(c) such that the sum of wire-lengths of all wires in the resulting decomposition (Figure 4(d)) is minimized.

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

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

4. We then iterate steps 2 and 3 over all Boolean nodes of the network until all nodes have at most two fanins.

5. Finally, we run global placement in incremental mode on the resulting network of primitive gates. The reason for running the global placement at this stage is to legalize the resulting placement. Since the core algorithm and net model of both the local 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 incremental global placement at most β times (where β is a user defined variable) during the technology decomposition algorithm. This is to ensure that our local placement runs utilize accurate node placement information at all times. We experimented with several values of β, and found that $\beta = 10$ resulted in 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 5.1 *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 local 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 $FI_s = (f_{s_1}, f_{s_2}, \ldots, f_{s_k})$ 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 4(c), the counter clockwise traversal gives the following circular order: $(f_1, f_2, f_3, f_4, f_5, f_6, f_7, f_8, f_1, \ldots)$. 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 4(c) is shown in Figure 5(a).

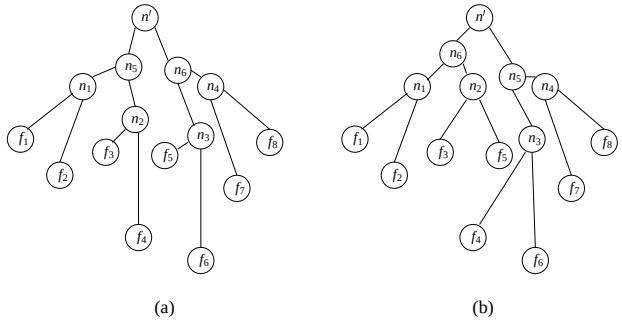

Figure 5: (a) Angle ordering solution. (b) Furthest pair ordering solution.

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 4(c) is shown in Figure 5(b).

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 5(a) and 5(b)).

5.2 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 [7].

We describe our area and delay minimization algorithms below:

5.2.1 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.

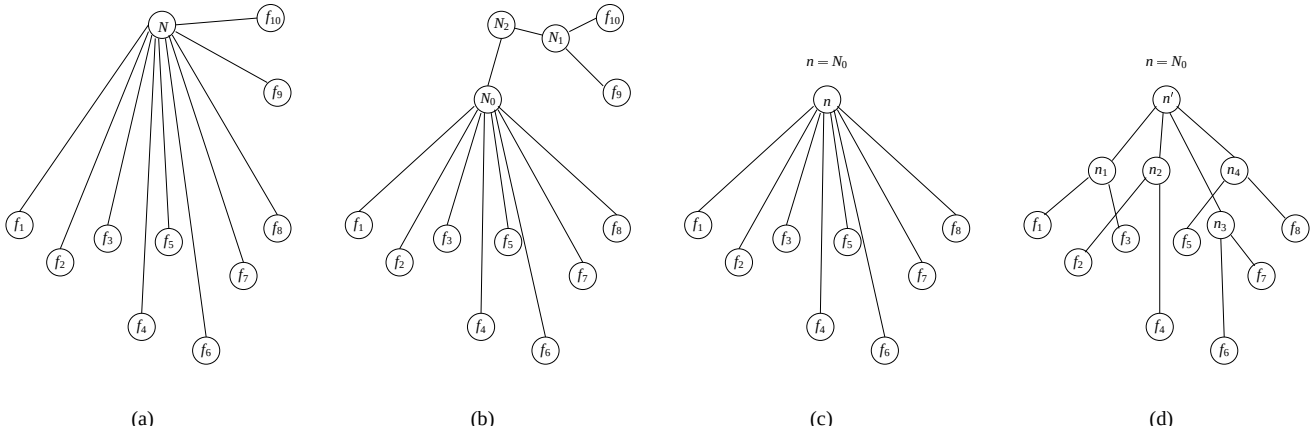

Figure 4: (a) Node N to be decomposed. (b) Decomposition of N into ANDs and ORs. (c) Node N_0 before two-step decomposition. (d) Node N_0 and its two-step decomposition solution.

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* $w_I(n_m)$ of match m as the total length of the fanin nets of n_m plus the fanin wire cost of all its fanins. Formally, $w_I(n_m) = \sum_{f_i \in FI} w_I(n_{f_i}) + l(n_{f_i})$, 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, $w_I(n_m)$ 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 $c(m) = a(m) + \alpha(w_I(m) + l(n))$, where α is a user-defined weighting variable.

Figure 6 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.

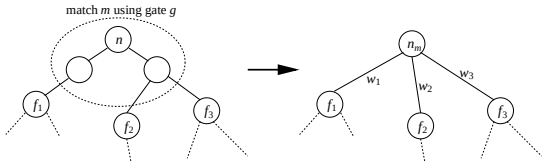

Figure 6: Definition of Cost Elements.

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 6). 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 local placement algorithm described in Section 3 to estimate the position of the new node.

After a certain number of nodes have been matched, the incremental 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 incremental global placement algorithm is called at most β times.

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

5.2.2 Delay Minimization

For delay minimization, we integrate the delay mapping algorithm developed by Touati [18] with global placement. Like area minimization, this algorithm decomposes the subject graph into trees and use dynamic programming to find a good mapping.

When considering a match m with gate g at node n, the arrival time of n is not computed. Rather, a piece-wise linear function f is computed. This is best explained by an example. Let g be a gate with 2 input pins. The delay model from both input pins of g is characterized by $\alpha(1)$, $\beta(1)$, and $\alpha(2)$, $\beta(2)$, where α is the intercept and β the slope of the delay model used in this paper, as shown in Figure 7(a) and (b). Assuming the arrival times at fanin 1 and 2 are $A(1)$ and $A(2)$ respectively, the maximum delay values for all load values are shown in Figure 7(c), which is a piece-wise linear function for all possible load values for match m [18]. Each match at n is therefore characterized by a piece-wise linear function. The best match at n is a piece-wise linear function which is computed by taking the minimum of the piece-wise linear functions of all matches at n. The best gate to implement n is selected only when the load at the output of n is known, which occurs during the backward traversal phase of the dynamic programming algorithm.

The local and global placement algorithms are interleaved with the delay mapping algorithm in a similar fashion as in the area mapping algorithm.

6 Experimental Results

The library we use is the MSU standard cell library [14]. We modified the library based on resistances and capacitances obtained from SPICE

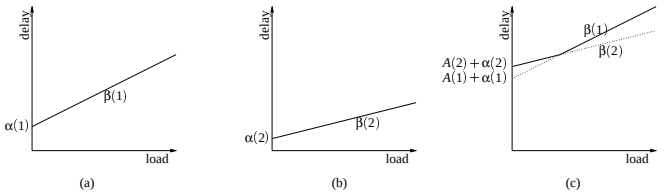

Figure 7: (a) Delay function of fanin 1. (b) Delay function of fanin 2. (c) Maximum delay function of fanin 1 and 2.

characterizations (using the 0.1μm 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.

Twenty-five of the large circuits in the 1991 MCNC benchmark set are selected for all experiments. Pin locations are determined uniformly at random. A small initial placement square is used for each circuit. The size is then expanded during routing to allow all nets to be successfully routed.

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 [14]. 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 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 2) 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. The 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 underestimates 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 3 reports the delay comparison between technology dependent optimization minimizing area using SIS versus our approach. In addition to the advantage of not having to estimate net lengths (and thereby solving the Timing Closure problem as shown in Table 2), our scheme exhibits a significant reduction in interconnect delay as seen from Table 3. For the example *dalu*, this translates into a 23% reduction in total delay. Using our scheme, the average reduction in interconnect delay is 12%, and the average reduction in total delay is 8%.

Table 2: 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%

The improvement in interconnect delay of our method is accompanied by a small penalty in active area, as shown in Table 3. The average penalty in active area is 10%.

Table 3: Results of area and wire-length minimization.

Name	Delay		Area
	Interconnect	Total	
C1908	−3%	−1%	2%
C2670	−32%	−13%	20%
C3540	−33%	−9%	11%
C432	22%	13%	1%
C499	17%	16%	3%
C6288	46%	0%	2%
C880	−1%	−0%	13%
apex6	−45%	−27%	9%
cht	−24%	−28%	3%
dalu	−36%	−23%	19%
example	−6%	−7%	11%
frg2	18%	26%	20%
i5	−27%	−31%	18%
i6	13%	3%	8%
i7	−12%	−12%	14%
i8	−1%	4%	15%
i9	−2%	−5%	2%
pair	−40%	−25%	14%
rot	−28%	−3%	6%
term1	−21%	−6%	−0%
ttt2	−8%	−2%	11%
vda	5%	−7%	6%
x1	−37%	−19%	8%
x3	−43%	−36%	11%
x4	−18%	−11%	15%
Ave	−12%	−8%	10%

Table 4 shows the delay comparison between technology mapping minimizing delay using SIS versus our approach which includes the wire-planned kernel extraction, wire-planned duplication, and delay optimization in technology dependent algorithms described above. In all experiments, γ is 20, i.e. the incremental global placement is called after every 20 kernels have been extracted. The duplication step is

applied after the technology decomposition step. Again, we achieve a significant reduction in interconnect delay and total delay. Table 4 also shows the area comparison of SIS and our delay minimization scheme. As seen from this table, we achieve a significant reduction in area in addition to the delay reduction. We save an average reduction of 18% in area, 24% in interconnect delay, and 18% in total delay. So, in addition to minimizing Timing Closure problems, our algorithm is able to reduce total circuit delay as well as total circuit area.

Table 4: Results of delay minimization.

	Delay		
Name	Interconnect	Total	Area
C1908	1%	3%	−10%
C2670	−19%	−7%	−21%
C3540	−15%	−13%	−14%
C432	−52%	−26%	4%
C499	16%	17%	−12%
C6288	4%	−5%	−9%
C880	−49%	−24%	−17%
apex6	−30%	−27%	−25%
cht	−52%	−45%	−36%
dalu	−12%	−13%	−18%
example2	−33%	−24%	−27%
frg2	−48%	−18%	−28%
i5	8%	−10%	−7%
i6	−24%	−8%	−33%
i7	−28%	−13%	−32%
i8	−32%	−24%	−19%
i9	−58%	−42%	−32%
pair	−20%	−3%	−10%
rot	−2%	9%	−11%
term1	−34%	−10%	−14%
ttt2	−33%	−21%	−14%
vda	−12%	−9%	−13%
x1	1%	6%	−14%
x3	−33%	−28%	−12%
x4	−41%	−35%	−26%
Ave	−24%	−15%	−18%

7 Conclusions

In this paper, we have presented an approach that addresses the timing closure problem in IC design. Our approach integrates both technology independent and technology dependent steps 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 local and incremental global placement algorithms to achieve this goal.

We have introduced wire-planned kernel extraction, wire-planned duplication, technology decomposition, and technology mapping algorithms using this integrated flow. Our technology dependent algorithms include both area and delay minimization.

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%.

• All placement algorithms (incremental global, full global, and local) 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 12% in interconnect delay, and about 8% in total circuit delay for area and wire-length minimization. This is accompanied by an area penalty of 10%.

• For delay optimization, we achieve an average reduction of about 24% in interconnect delay, and about 15% in total circuit delay. In addition to this, area is reduced by 18% on average.

References

[1] Cadence Design Systems, Inc., 555 River Oaks Parkway, San Jose, CA 95134. *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, Aug 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] W. Gosti, A. Narayan, R. K. Brayton, and A. L. Sangiovanni-Vincentelli. Wireplanning in logic synthesis. In *ICCAD*, pages 26–33, Nov 1998.

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

[8] S Khatri, A Mehrotra, R Brayton, A Sangiovanni-Vincentelli, and R Otten. A novel VLSI layout fabric for deep sub-micron applications. In *DAC*, New Orleans, June 1999.

[9] 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.

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

[11] 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.

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

[13] M. Pedram and N. Bhat. Layout driven technology mapping. In *DAC*, pages 99–105, June 1991.

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

[15] 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 *ICCD*, pages 250–257, October 1999.

[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*, Napa Valley, CA, 1997.

[18] Hervé J. Touati. *Performance-Oriented Technology Mapping*. PhD thesis, University of California Berkeley, Univ. of CA, Berkeley, November 1990. Memorandum No. UCB/ERL M90/109.

[19] 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.