Combining Technology Mapping with Layout

Massoud Pedram
Department of EE - Systems
University of Southern California
Los Angeles, CA 90089

Narasimha Bhat
AT&T Bell Labs
600 Mountain Avenue
Murray Hill, NJ 07974

Ernest S. Kuh
Department of EECS
University of California
Berkeley, CA 94720

Contents

List of Figures

List of Tables

ABSTRACT

Due to the significant contribution of interconnect to the area and speed of today's circuits and the technological trend toward smaller and faster gates which will make the effects of interconnect even more substantial, interconnect optimization must be performed during all phases of the design. The premise of this paper is that by increasing the interaction between logic synthesis and physical design, circuits with smaller area and interconnection length, and improved performance and routability can be obtained compared to when the two processes are done separately. In particular, this paper describes an integrated approach to technology mapping and physical design which finds solutions in both domains of design representation simultaneously and interactively. The two processes are performed in lock-step: technology mapping takes advantage of detailed information about the interconnect delays and the layout cost of various optimization alternatives; placement itself is guided by the evolving logic structure and accurate path-based delay traces. Using these techniques, circuits with smaller area and higher performance have been synthesized.

Key Words: **Technology mapping, Logic decomposition, Placement, Layout-driven logic synthesis.**

1 INTRODUCTION

A principal goal of the electronic design automation effort is to provide designers with the capability to employ a complete top-down design methodology, compiling abstract architectural descriptions into an optimal implementation for a specific physical media. In fact, as density increases, system and ASIC designers have no choice but to converge to such a top-down design methodology. However, there is at least one major difficulty that must be overcome in order to develop the full potential and promise of this methodology. The problem is that the top-down design approach makes high-level decisions about optimization, scheduling, allocation, logic partitioning and restructuring, and so forth in terms of abstract views of behavior and structure. During these early steps, factors such as characteristics of the physical media, layout, interconnect and parasitics are ignored. Physical design which is expected to address these issues, comes much later in the design hierarchy. By then, many of the key architectural and structural decisions have been made, hence, limiting the capability of the physical design tools to generate the "best" solutions in terms of area and performance. In addition, once some parameter at the physical design stage fails to satisfy a constraint imposed on it, the synthesis must be modified (even repeated) so as to accommodate the constraint. The new change may cause some other constraint to be violated and the process must be iterated.

It is, therefore, necessary to develop models, algorithms and techniques to increase the power and robustness of the top-down design methodologies. The goal – which is easy to state but difficult to achieve – is to *integrate* the various design steps keeping the computational complexity manageable. This paper is a step toward achieving this objective, that is, it describes novel techniques to couple logic synthesis to physical design and to explicitly control the interconnection length during logic synthesis. This will allow physical design and logic synthesis to evolve together and will ensure higher degrees of system level integration.

The goal of logic synthesis is to produce a circuit which satisfies a set of logic equations, occupies minimal silicon area and meets the timing constraints. Logic synthesis is often divided into a technology-independent and a technology-dependent phase. In the first phase, transformations are applied on a Boolean network to find a representation with the least number of literals in the factored form. Additional timing optimization transformations are applied on this minimal

[1]This work was sponsored in part by the National Science Foundation and by the Semiconductor Research Corporation.

area network to improve the circuit performance. The role of the technology-dependent phase is to finish the synthesis of the circuit by performing the final gate selection from a target library. The technology-dependent phase is, to a large extent, constrained by the structure of the optimized Boolean network. Most previous work in logic synthesis [4, 3] has focused on minimizing gate area and delay through a chain of gates without considering the area needed to hold the interconnect lines or the delay through the lines. It is generally assumed that interconnect optimization can be relegated to the physical design phase. Only recently some attention has been given to the interconnect optimization during logic synthesis [1, 20, 19].

1.1 Interconnect Effects

Interconnections are becoming a major concern in today's high-performance, high-density ASIC designs because the distributed RC time delay of these lines increases rapidly as chip sizes grow and minimum feature sizes shrink [2]. With recent studies [23, 10] indicating that interconnections occupy more than half the total chip area and account for a significant part of the chip delay, it is appropriate that wiring is integrated into the cost function for logic synthesis. To elaborate on this point, consider Figure 1 which shows a performance-optimized two-input NAND gate driving a performance-optimized inverter gate through 0.2 cm of aluminum interconnect (2 μm wide, 0.5 μm thick, with a 1.0 μm thick field oxide beneath it). 0.2 cm is the expected length of a *local* interconnect line on a $2cm \times 2cm$ chip [2]. Two methods are used to calculate the rise time (to 50% of its final value) at the input of the inverter gate: one method ignores the capacitance and resistance of the interconnect line, the second method accounts for them [22]. Gate delays are taken from data sheets for an industrial 1-micron ASIC library; interconnect capacitance and resistance are calculated using expressions given in [2]. The delay calculations clearly show that interconnect capacitance dominates gate input capacitance and interconnect resistance may be ignored without introducing much error. When the RC tree forms branches, delays for the branching nodes can be calculated independently and accumulated to obtain the delays at the sink nodes. This calculation, however, requires knowledge of net topologies which is not available before global routing. This effect will not be addressed here. Furthermore, the transmission line properties of interconnect lines are ignored for on-chip connections. Therefore, an accurate expression for propagation delay through gates connected by local interconnect lines is given by

$$d(z) = \tau + R_s(C_z + Cl)$$

where τ is the intrinsic gate delay, R_s is the on-resistance of the driver gate, C_z is the input capacitance of the fanout gate, C is the interconnect capacitance per unit length and l is the interconnect length.

Figure 1 goes here.

In summary, with the existing technology, the capacitive term is dominated by the capacitance between the interconnection and substrate. For local aluminum lines, the resistive term is dominated by the on-resistance of the MOS transistor; for polysilicon and global aluminum lines on large-size circuits, the resistive term is controlled by the interconnection resistance. As the chip dimension increases and the minimum feature size decreases, the interconnection resistance increases rapidly while the MOS on-resistance remains relatively unchanged; the interconnection capacitance bottoms at about 1 - 2 pF/cm while the input gate capacitance decreases. Therefore, the distributed RC delay of interconnect lines will become even more dominant in the future.

1.2 Concepts and Examples

A *Boolean network* $\mathcal{N}$, is a directed acyclic graph (DAG) such that for each node in $\mathcal{N}$ there is an associated representation of a Boolean function f_i, and a Boolean variable y_i, where $y_i = f_i$. There is a directed edge (i, j) from y_i to y_j if f_j depends explicitly on y_i or y_i'. A node y_i is a *fanin* of a node y_j if there is a directed edge (i, j) and a *fanout* if there is a directed edge (j, i). A node y_i is a *transitive fanin* of a node y_j if there is a directed path from y_i to y_j and a *transitive fanout* if there is a directed path from y_j to y_i. *Primary inputs* are inputs of the Boolean network and *primary outputs* are its outputs. *Intermediate nodes* of the Boolean network have at least one fanin and one fanout. A Boolean network is an implementation or representation of a set of incompletely specified Boolean functions.

Figure 2 goes here.

Given a Boolean network optimized by technology-independent logic operations and a target library, *technology mapping* is the process of binding nodes in the network to gates in the library such that the area of the final implementation is minimized, and timing constraints are satisfied. We illustrate the incorporation of the interconnect into technology mapping with a simple example. Figure 2a shows a small portion of a NAND-decomposed Boolean network. Source nodes s_i have either been mapped (and hence have been assigned matching gates and positions) or are fixed at the circuit boundary. Note that s_1 and s_2 are positioned near one another but far from s_3 and s_4. The objective is to transfer the signals from s_i's to the sink node t implementing the desired logic function while using minimum wire length. Conventional technology mappers attempt to find a solution with the smallest area gate which matches as many intermediate nodes as possible (the solution with one AND4 gate in Figure 2a). This is a good approach if the fanin gates s_i can be placed near the matching gates. However, in many cases, these gates are either strongly connected to different gate clusters on the layout plane or are fixed at the circuit boundary and hence may have positions far from one another and from the matching gate. Therefore, a solution with one distribution point may incur a large interconnection cost. In fact, there is often an optimum number of mapped gates greater than one which will result in overall minimum wire cost as depicted in Figure 2. If the number of sources is large, say four or more, then it will pay off to consider how close the sources can be placed by a good placement optimizer before deciding whether a solution of one gate (with high fanin count) or a solution of more than one gate (with low fanin counts) should be chosen. The technology mapper proposed here selects the solution with one AND2 and one AND3 gate in Figure 2a.

Figure 2b illustrates the importance of a layout-directed technology decomposition for the technology mapping scheme. This figure shows the same decomposition tree as in Figure 2a. However, this time as a result of placing the NAND-decomposed network, source nodes s_1 and s_3 (s_2 and s_4) have been positioned near one another. Signals coming from s_1 and s_3 (s_2 and s_4) enter the network at topologically distant points. This is undesirable because the mapper has lost the option of reducing the wiring cost by breaking one big gate into smaller gates, i.e., in Figure 2b, the mapping solution with one AND4 gate is superior to other solutions in terms of both total gate area and interconnection length. Therefore, Figure 2a provides better technology decomposition

(and hence potential for higher quality mapping) than that in Figure 2b.

1.3 Overview

In this paper, we present LDTM [20], a technology mapping program, built on top of MIS [4], which tightly and interactively couples mapping to placement. LDTM's key idea is to generate a placement of the optimized multi-level Boolean network which captures the structure of the network. The placement information is used to evaluate the cost of a gate matching during decomposition and mapping processes. The placement is dynamically updated in order to maintain the correspondence between logic and layout representations. In the end, a mapped network along with a *companion* placement solution are generated. The placement solution is then globally relaxed in order to produce a feasible placement according to the target layout style (e.g., standard-cell or sea-of-gates). This design flow is in contrast with the existing synthesis systems which separate technology mapping and placement steps.

Technology Mapping is driven by layout information derived from the placement of the Boolean network. It is, therefore, essential to generate a placement solution which not only captures the global connectivity structure of the network, but also produces the shortest directed path between any pair of primary input – primary output nodes. [16] describes a flow-oriented approach to the placement of general directed acyclic graphs.

In Section 2, the technology mapping paradigm is presented. Section 3 describes a technique for creating a feasible placement solution from the companion placement solution. Sections 4 and 5 contain experimental results, discussions and future research directions.

2 LAYOUT-DRIVEN TECHNOLOGY MAPPING

A successful and efficient solution to the technology mapping problem was suggested by K. Keutzer and implemented in DAGON [12] and MIS [9]. The idea is to reduce technology mapping to DAG covering and to approximate DAG covering by a sequence of tree coverings which can be performed optimally using dynamic programming. DAGON and MIS technology mappers generate circuits with small active cell area but ignore area contributed by interconnections between gates. Consequently, these mappers produce gates with high fanin count which often increase routing congestion during the final layout and increase interconnection lengths. Similarly, performance-oriented tech-

nology mapping programs suffer from lack of detailed information about the interconnect delay. The approach presented here, however, explicitly considers interconnect and routing complexity during the mapping.

Our layout-directed technology mapping is based on the DAG covering formulation which can be summarized as follows. A set of complete base functions is chosen, such as a two-input NAND gate and an inverter. The optimized logic equations (obtained from technology independent optimization) are converted into a graph where every node is one of the base functions. This graph is called the *subject graph*. Each library gate is also represented by a graph consisting of only base functions. Each such graph is called a *pattern graph*. A library gate may have many different pattern graphs. The technology mapping problem is then defined as the problem of finding a minimum cost covering of the subject graph by choosing from the collection of pattern graphs for all gates in the library. For area optimization, the cost of a cover is defined as the sum of gate areas. For performance optimization, the cost of a cover is defined as the critical path delay of the resulting circuit. In particular, the mapper binds a given logic circuit onto a set of gates in the target library minimizing *post–layout* area, delay or area under delay constraints.

2.1 Technology Decomposition

The procedure for converting an optimized Boolean network into the subject graph (i.e., *technology decomposition*) is not unique, and it is an open problem to determine which of the possible subject graphs yields an optimum solution when an optimum covering algorithm is applied [5]. The goal of our technology decomposition procedure is to find a circuit representation with minimum signal arrival time at the primary outputs and minimum number of wire crossings (given an initial placement of the optimized Boolean network).

The decomposition process starts by constructing AND-OR trees implementing the sum-of-product representation of the logic function associated with each intermediate node in the Boolean network. The function of AND subtrees is to compute the product terms (cubes) and that of the OR subtrees is to compute the sum of the product terms. The input signals to the AND subtrees and then the cubes in the OR subtrees are ordered. The conversion from the ordered AND-OR subtrees to the gates in base function set is accomplished using unbalanced NAND decomposition by providing late arriving inputs with shorter paths through the NAND-decomposed subnetwork

[25, 17].

Figure 3 goes here.

In order to derive the input signal ordering, one refers to the companion placement solution for the Boolean network. Each multi-pin net signal is modeled by a star connection from the source toward the sinks. By circularly traversing around each node (for example, starting from the positive horizontal axis and proceeding in a counter-clockwise fashion), a unique ordering of the input signals to the node can be determined. This ordering is directly related to the positions of the fanin nodes with respect to the node in question.

The cube ordering is achieved by setting up a linear assignment problem. S slots are placed on an imaginary inner circle around the node, and the projections of the fanin signals into an imaginary outer circle around the node are found (Figure 3). Then, a linear assignment cost matrix C is set up whose c_{ik} entry corresponds to the cost of assigning cube i to slot k. This entry is equal to zero if slot k falls inside the shortest circular span for the immediate support of cube i. Otherwise, the cost is proportional to the angular distance of slot k from the nearest end of the support span of cube i. The linear assignment program [7] determines a cube assignment with the minimum sum-cost. The cube ordering is easily derived from the cube positions obtained by the above linear assignment procedure. The process of ordering input signals, cubes and then primitive gate decomposition is recursively applied to all nodes in the Boolean network in order to produce the subject graph.

2.2 DAG Covering for Minimum Layout Area

Consider a Boolean network N, which has been transformed into a subject graph consisting of only two-input NAND and inverter gates. This is the network in its unmapped form which will be referred to as the *inchoate* network, $N_{inchoate}$. In DAGON, $N_{inchoate}$ is partitioned into a set of maximal *tree*s, T_i, and an optimal dynamic programming solution is found for each tree. In MIS, $N_{inchoate}$ is split into a set of *logic cone*s K_i, where each cone corresponds to a primary output and all its transitive fanin nodes. This allows covering across tree boundaries and, as a result, may duplicate logic. The MIS technology mapper implements DAGON as a subset. Our mapper uses

cone partitioning.

Figure 4 goes here.

Assume that the cost of match m at node v is to be calculated (Figure 4). This cost consists of two (linearly combined) terms:

$$area\_cost(v,m) = area(gate(m)) + \sum_{v_i \in inputs(v,m)} area\_cost(v_i, m_i)$$

$$wire\_cost(v,m) = wire(gate(m), gate(v_i)) + \sum_{v_i \in inputs(v,m)} wire\_cost(v_i, m_i).$$

where $inputs(v,m)$ refers to the list of nodes of N which correspond to the inputs of m. $gate(m)$ is the physical gate corresponding to m. $gate(v_i)$ is the best gate matching at node v_i. The area cost calculation is similar to that in MIS. The wire cost $wire\_cost(v,m)$ consists of two terms. The first term is the interconnection length required to complete connections from $gate(m)$ to its fanin gates, i.e., $gate(v_i)$. The second term is the dynamic programming recursive cost and represents the sum of wire lengths required to connect all gates from primary inputs up to $gate(v_i)$. In order to calculate $wire\_cost(v,m)$, the position of m must be known as shown next.

At the beginning, nodes are assigned valid *placePosition*s based on the initial global placement solution. As nodes are mapped, *mapPosition*s are calculated and stored on nodes. In particular, match m is placed at the center of mass (or median) of its fanin and fanout rectangles. Due to the postorder traversal of the network during the mapping procedure, $inputs(v,m)$ have already been mapped and therefore their *mapPosition*s are used; $outputs(v)$ are not mapped yet and their *placePosition*s are used.

Figure 5 goes here.

Depending on the wire length metric adopted, the local placement problem can be solved efficiently or can become difficult. Consider Figure 5, which shows the enclosing rectangles for

the fanin and fanout nets of match m at v.[1] Given a norm and the coordinates of these fanin and fanout rectangles r, the problem is to find a point p which results in the minimum sum of distances between that point and the rectangles. In case of the Manhattan norm, the solution easily follows by observing that the distance function has a separable form with respect to the variables x and y. That is, the x distance of point p from rectangle r is $\frac{1}{2}(|r.ll.x - p.x| + |r.ur.x - p.x| - |r.ur.x - r.ll.x|)$ where *ll* and *ur* refer to the lower left and the upper right of rectangle *r*. The constant term is dropped and the problem can be restated as: Find x such that $\sum_i |x_i - x|$ is minimum where x_i corresponds to either the left or the right corner point coordinates of each of the rectangles. The problem is a special case of solving for the median of a graph which is presented in [11]. It can be shown that this problem, treating only a linear tree rather than a general graph, is very easy to solve; the solution is the median point for the sorted list of x_i's. For the Euclidean norm, the optimal point location problem can be solved approximately by placing m at the center of mass of its fanin and fanout rectangles [18].

When constructing the enclosing rectangle of fanin net i, it is important to know fanout nodes of source node v_i. These fanout nodes of v_i are dynamically defined based on the current partially mapped network. First, we give some definitions. A *sink* node in a pattern graph is defined as a node which does not fanout to any other node in the pattern graph. In Figure 4, assume that cone K_1 has been mapped. At this point, nodes can be classified into four categories. An *egg* is a node which has not been processed (visited) by the mapper. A *nestling* is a node in the current cone, K_2, which has been visited. It cannot be predicted whether a *nestling* will be present in the final mapped network until primary output PO_2 is reached. A *dove* is a node in K_1 which is a non-sink element of some pattern match. Such a node will not be present in the final mapped network because it has been merged into another. A *hawk* is a node in K_1 which is a sink node in some pattern match. Such a node will inevitably show up in the final mapped network. Note that every *dove* has been merged into (fallen prey to) at least one *hawk*. A *nestling* can become a *hawk* or a *dove*. Due to the possibility of logic duplication, it may be possible for a *dove* to reincarnate and restart the node's life cycle as an *egg* and later become a *hawk*. The *dynamic fanout* of fanin v_i for match m at v is a *hawk*, a *nestling* or an *egg* which has v_i as its fanin. For example, the list of dynamic fanouts of node v_4 consists of nodes v, x_2 and f_4. During the construction of the enclosing rectangle of net i (say 4), nodes covered by the current match are excluded from the net. In addition, *mapPosition*s

[1]The enclosing rectangle for a fanin (fanout) net is the smallest rectangle enclosing the gates connected by the net.

are used for *hawk*s and *nestling*s (x_2 and v_4) and *placePosition*s are used for *egg*s (f_4).

After positioning $gate(m)$, the wire cost associated with the matching of m at node v must be calculated. This cost consists of the sum of the wire lengths from m to its fanins. Consider fanin v_i of m. If it is driving only the input pin of m, the wire length calculation is point-to-point. However, if it is driving multiple fanout pins (including input pin of m), then the half perimeter length of the minimum box bounding all pins on the net is used. Note that during technology mapping for minimum area, only length of the line connecting each fanin v_i to m is of interest. Therefore, the net length must be divided by the dynamic fanout count at v_i in order to get the expected wire length contributed by connection from v_i to m and thereby avoid duplicate accounting of the wire cost.

2.3 DAG Covering for Minimum Circuit Delay

In the delay mode, the best mapping at a node is determined based on the arrival time of the signal at the node output. As pointed out earlier, it is the delay in the interconnect that is of prime importance. Hence, it is only natural that wiring delay is incorporated into the calculation of the arrival time during technology mapping.

Consider a gate g with output line y and input lines i, $i = 1 \cdots p$. Let g fanout to inputs of g_j. In a simple linear delay model, the delay through g is a linear function of its output load capacitance C_L. The slope of this linearity can be thought of as the *output resistance* and the offset (at zero C_L) can be thought of as the *intrinsic delay* through g. In general, the delays from different inputs to the output are different. Therefore, the intrinsic delay from input i to y is denoted by I_i, and the output resistance at y corresponding to input i is denoted by R_i. I_i and R_i each have separate values for rising and falling delays.

Based on this general model, the arrival time at y from input i, t_{y_i}, can be easily calculated as $t_{y_i} = t_i + I_i + R_i\, C_L$ where t_i is the arrival time at input line i. Using a worst case analysis, the *output arrival time* at y, t_y is defined as the time at which all signals from input lines i will be available at y and is given by $t_y = max\{t_{y_i}\}$ computed over all $i, i = 1 \cdots p$. This calculation for the arrival time requires that the value of C_L be known. C_L is the equivalent capacitive load at y. This capacitance is modeled as $C_L = \sum_{j=1}^{n} C_j + C_w$ where C_j denotes the capacitance at the input of fanout gate g_j, and n is the number of fanout nodes. C_w represents the capacitance due to the interconnections which connect g to its fanout nodes. We calculate C_w accurately using

the placement information which provide the horizontal and vertical extents of the signal nets and using a technology file which provides the capacitance per unit length of horizontal and vertical interconnect.

In Figure 4, fanouts of v are not yet mapped. This implies that the load C_L, at the output of *gate(m)* cannot be determined exactly. This difficulty can be handled by assuming a *default load*, i.e., all types of gates are assumed to have the same input capacitance. This assumption is also adopted in MIS2.1. However, in order to calculate the wiring capacitance, positions of gates at the fanouts of v must be known. This information is not available and instead positions of the fanout gates are read from the initial placement solution of the subject graph. The simplification gives rise to inaccuracies in the arrival time calculation. To prevent the inaccuracy from propagating through, the following observation is used: When matching m at v, the capacitance at the output of *inputs(v,m)* is known because the type and position of their fanout gate, which is *gate(m)*, is known. If the output arrival times of *inputs(v,m)* are updated, then the input arrival time of *gate(m)* is accurate. Therefore, the output arrival time for $gate(m)$ can be calculated with less error (that is, error will be due to the unknown load only).

3 PLACEMENT RELAXATION

After the logic synthesis stage, a net list of gates and a companion placement solution are available. The placement solution, however, has overlapping gates and has not yet been mapped to rows (in the case of the standard cell layout methodology) or to slots (in the case of the sea-of-gates style). The objective of global relaxation step is to eliminate gate overlaps and produce an even distribution of gates over the layout image. Two basic approaches are generally used for mapping a global placement result to legal locations: (1) Perform a minimum squared error linear assignment which maps the cells in the global placement to the legal positions simultaneously; (2) Use a hierarchical bi-partitioning technique to obtain a feasible placement solution.

In our application, the initial (globally optimized) placement is modified throughout the synthesis (based on local considerations only), and therefore, the resulting companion placement is not globally optimized. However, we do not want to throw away the companion placement (which has influenced many of our decisions during the synthesis) and place the mapped network from scratch.

We have adopted the top-down bi-partitioning heuristic in the following way. Our placement

procedure consists of alternating and interacting global function optimization and partitioning steps. In particular, for a circuit with M gates, the placement procedure goes through $m = \lceil log_2 M \rceil$ steps in order to produce a detailed placement. Now, assume that an initial placement solution for the circuit and two parameters N_s and N_f are given. These parameters specify the start and finish conditions for the relaxation procedure, that is, relaxation begins when number of modules per hierarchical region is N_s and ends when this number is N_f. Let $s = \lceil log_2 N_s \rceil$, $t = \lceil log_2 N_f \rceil$, then $m \geq s \geq t \geq 0$. We modify the placement procedure so that it goes through steps $m - s, \cdots, m - t$ only, thereby, achieving the relaxation goal without drastically disturbing the initial placement solution. Note that $N_s = M$ corresponds to doing placement from scratch while $N_f = 1$ corresponds to the detailed placement (one module per region). We have obtained the best results when setting $N_s = M/8$ and $N_f = 1$.

4 EXPERIMENTAL RESULTS

Our objective was to show that by integrating technology mapping and gate placement, one can improve the quality of mapping both in terms of layout area and circuit performance. In order to provide a fair basis for comparison, two pipelines were used to produce the results: 1) Read in the optimized circuit; do balanced tree technology decomposition; read in the *lib2.genlib* standard cell library; run MIS technology mapper in timing mode; write the mapped circuit to the database; do detailed placement and routing. 2) Read in the optimized circuit; do layout driven technology decomposition; read in the *lib2.genlib* standard cell library; run LDTM in timing mode; do detailed placement and routing. The following tools were used for generating the layouts: the GORDIAN package for placement [14], the TIMBERWOLF global router [15], and the YACR detailed router [21].

Table 1 goes here.

In both cases, the technology-independent optimizations were performed using the MIS program. The benchmarks were optimized for minimum area using the *rugged script* [24]. This script

produces the optimized circuits. The literal count results (before technology mapping) are listed in Table 1.

Table 2 goes here.

Table 2 shows *post-mapping* comparisons between the MIS and LDTM in terms of total gate area and logic delay calculated by ignoring the wiring loads (i.e., using the input capacitances, the intrinsic and fanout delays for logic gates only), and the cpu times on a Sparc Station II workstation. The MIS mapper produces somewhat better results in terms of logic delay and is faster by a factor of 1.8.

Table 3 goes here.

Table 3 shows *post-placement* comparisons between the MIS and LDTM in terms of total chip area and circuit delay calculated after placement and routing and accounting for the the wiring loads. As expected, LDTM shows an average chip area improvement of 7% and a total delay improvement of 9% compared to MIS. This improvement is mainly due to reduced wiring load on critical signal paths as a result of LDTM's gate selection policy. We used a value of 3 pF/cm for the capacitance per unit length. As the technology scales down and the contribution of the wiring delay to total circuit delay increases, the percentage improvement of a layout-driven mapper (such as LDTM) over a conventional mapper (such as that of MIS) will increase.

These tables were generated by running the MIS and LDTM mappers in the timing mode. A similar trend existed when we ran these mappers in the area mode.

5 CONCLUDING REMARKS

In this paper, we put forth techniques for coupling logic synthesis and placement. To achieve this objective, we studied the effects of interconnect on circuit area and performance, presented

appropriate models and computational procedures for estimating wiring delay during synthesis, and introduced a scheme for maintaining simultaneous and interactive data representations in logic and layout domains.

Layout information should be considered during all logic synthesis operations in order to improve the circuit routability and reduce the interconnect contribution to circuit area and delay. This is important since, as discussed earlier, interconnects play a primary role in determining the longest paths through a circuit. Therefore, layout-driven algorithms and techniques for performing the various logic operations must be developed.

In manipulating the initial representation of the logic function, five operations are key: decomposition, extraction, factoring, substitution, and collapsing [5]. We are developing layout-integrated procedures for these logic operation which aim at minimizing the total interconnection length of the synthesized network as well as the total number of literals in the factored form representation of the network. For example, consider the extraction procedure which identifies common subexpressions among various functions. The literal value of a kernel measures the difference in the number of literals in the network if that rectangle is extracted and made into a new node [6, 4]. We can similarly define the interconnection value of a kernel as the difference in the total wire length in the network if that kernel is extracted and made into a new node. We then use a kernel-selection policy which chooses a kernel with the greatest cost reduction in terms of a linear combination of literal and interconnection values [19].

Performance optimization logic restructuring operations (e.g., depth reduction [13], partial collapse and resynthesis along the critical paths [25], logic clustering and partial collapse [26], etc) are often used to speed the synthesized network. Layout information can be exploited by most of these transformations. For example, consider circuit speed-up procedure given in [25]. This procedure identifies an ϵ-network (i.e., a sub-network in which all the signals have a slack within ϵ of the most negative slack), partially collapses the nodes in the ϵ-network, and redecomposes these nodes using a timing-driven decomposition scheme such that the resulting network is faster, and the area increase is minimal. During this process, if information about the interconnect delay becomes available (through a placement of the original network and incremental calculation of positions for the collapsed and later newly created nodes), then the following improvements are possible: the timing analysis for finding the ϵ-critical paths is performed more accurately; during decomposition both literal saving and interconnect saving values of the candidate divisors are considered; interconnect delays influence the selection of the best timing-divisors; and NAND-

decomposition of collapsed nodes can be performed using information about positions of fanin nodes.

The layout-driven technology mapping procedure can be easily extended to solve the problem of minimizing gate area plus wiring subject to required time constraints on the primary outputs. The idea is to incorporate the wiring area and delay into the initial calculation of area-delay trade-off curves and the subsequent gate selection steps [8].

Previous approaches for Table Look-Up (TLU) based FPGA's have aimed at minimizing the number of TLU's. However, routing resources in these architectures are very limited and therefore mapping for improved routability is an important consideration. The layout-driven approach can be extended to the FPGA synthesis problem.

References

[1] P. Abouzeid, K. Sakouti, G. Saucier, and F. Poirot. Multilevel synthesis minimizing the routing factor. In *Proceedings of the 27th Design Automation Conference*, pages 365–368, June 1990.

[2] H. B. Bakoglu. *Circuits, interconnections, and packaging for VLSI*. Addison-Wesley, 1990.

[3] K. Bartlett, D. Bostick, G. Hachtel, R. Jacoby, M. Lightner, P. Moceyunas, C. Morrison, and D. Ravenscroft. BOLD: A multiple-level logic optimization system. In *Proceedings of the* IEEE *International Conference on Computer Aided Design*, 1987.

[4] R. Brayton, R. Rudell, A. Sangiovanni-Vincentelli, and A. Wang. MIS: A multiple-level logic optimization system. IEEE *Transactions on Computer-Aided Design of Integrated Circuits and Systems*, CAD-6(6), Nov. 1987.

[5] R. K. Brayton, G. D. Hachtel, and A. L. Sangiovanni-Vincentelli. Multilevel logic synthesis. In *Proceedings of the* IEEE, volume 78, pages 264–300, February 1990.

[6] R.K. Brayton. Algorithms for multilevel logic synthesis and optimization. In P. Antognetti, G. DeMicheli, and A. Sangiovanni-Vincentelli, editors, *NATO ASI on Logic Synthesis and Silicon Compilation for VLSI*. Kluwer, Dordrecht, The Netherlands, 1987.

[7] R. E. Burkhard and U. Derigs. *Assignment and matching problems: solution methods with Fortran programs*. Springer Verlag, 1980.

[8] K. Chaudhary and M. Pedram. A near-optimal algorithm for technology mapping minimizing area under delay constraints. Proceedings of the 29th Design Automation Conference, June 1992.

[9] E. Detjens, G. Gannot, R. Rudell, A. Sangiovanni-Vincentelli, and A. Wang. Technology mapping in MIS. In *Proceedings of the* IEEE *International Conference on Computer Aided Design*, pages 116–119, November 1987.

[10] Y. A. El-Mansy and W. M. Siu. MOS technology advances. In G. Rabbat, editor, *Handbook of Advanced Semiconductor Technology and Computer Systems*, pages 229–259. Van Nostrand-Reinhold, Princeton, New Jersey, 1988.

[11] S. L. Hakimi. Optimum locations of switching centers and the absolute centers and medians of a graph. *Operations Research*, 12:450–459, 1964.

[12] K. Keutzer. DAGON: Technology mapping and local optimization. In *Proceedings of the Design Automation Conference*, pages 341–347, June 1987.

[13] K. Keutzer and M. Vancura. Timing optimization in a logic synthesis system. In *Proceedings of the International Workshop on Logic and Architectural Synthesis*, May 1987.

[14] J. M. Kleinhans, G. Sigl, F. M. Johannes, and K. J. Antreich. GORDIAN: VLSI placement by quadratic programming and slicing optimization. *IEEE Transactions on Computer-Aided Design*, CAD-10:356–365, March 1991.

[15] K. W. Lee and C. Sechen. A new global router for row-based layout. In *Proceedings of the* IEEE *International Conference on Computer Aided Design*, pages 180–183, November 1988.

[16] S. Mayrhofer, M. Pedram, and U. Lauther. A flow-based approach to the placement of boolean networks. In *IFIP International Conference on Very Large Scale Integration*, 1991.

[17] C. R. Morrison, R. M. Jacoby, and G. D. Hachtel. TECHMAP: Technology mapping with delay and area optimization. In *Proceedings of the International Workshop on Logic and Architecture Synthesis for Silicon Compilers*, pages 53–64, Grenoble, France, May 1988.

[18] M. Pedram. *An integrated approach to logic synthesis and physical design*. PhD thesis, University of California, Berkeley, August 1991. Technical Report UCB/ERL M91/69.

[19] M. Pedram and N. Bhat. Layout driven logic restructuring / decomposition. In *Proceedings of the* IEEE *International Conference on Computer Aided Design*, pages 134–137, November 1991.

[20] M. Pedram and N. Bhat. Layout driven technology mapping. In *Proceedings of the 28th Design Automation Conference*, pages 99–105, June 1991.

[21] J. Reed, A. Sangiovanni-Vincentelli, and M. Santamauro. A new symbolic channel router: YACR2. *IEEE Trans. on Computer-Aided Design*, 4(3):208–219, March 1985.

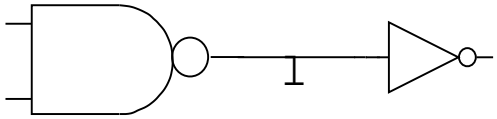
[22] T. Sakurai. Approximation of wiring delays in MOSFET LSI. IEEE *Journal of Solid State Circuits*, SC-18(4):418–426, August 1983.

[23] K. C. Saraswat and F. Mohammadi. Effect of scaling of interconnections on the time delay of VLSI circuits. IEEE *Transactions on Electron Devices*, ED-29:645–650, 1982.

[24] H. Savoj and H. Y. Wang. Improved scripts in MIS-II for logic minimization of combinational circuits. In *Proceedings of the International Workshop on Logic Synthesis*, May 1991.

[25] K. J. Singh, A. Wang, R. K. Brayton, and A. Sangiovanni-Vincentelli. Timing optimization of combinational logic. In *Proceedings of the* IEEE *International Conference on Computer Aided Design*, pages 282–285, November 1988.

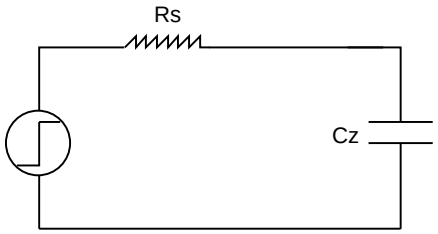
[26] H. J. Touati, H. Savoj, and R. K. Brayton. Delay optimization of combinational logic circuits by clustering and partial collapsing. In *Proceedings of the* IEEE *International Conference on Computer Aided Design*, pages 188–191, November 1991.



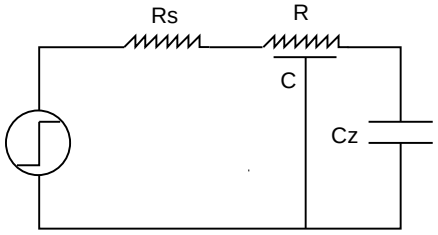
t = 0.3 ns R = 300 ohms/cm

Rs = 1 K-ohms C = 3.0 pF/cm

Cz = 0.1 pF l = 0.2 cm



d(z) = t + Rs Cz = 0.4 ns



d(z) = t + Rs Cz + Rs C l = 1.0 ns

Figure 1: Comparing the interconnect delay to gate delay for 1-micron ASIC library

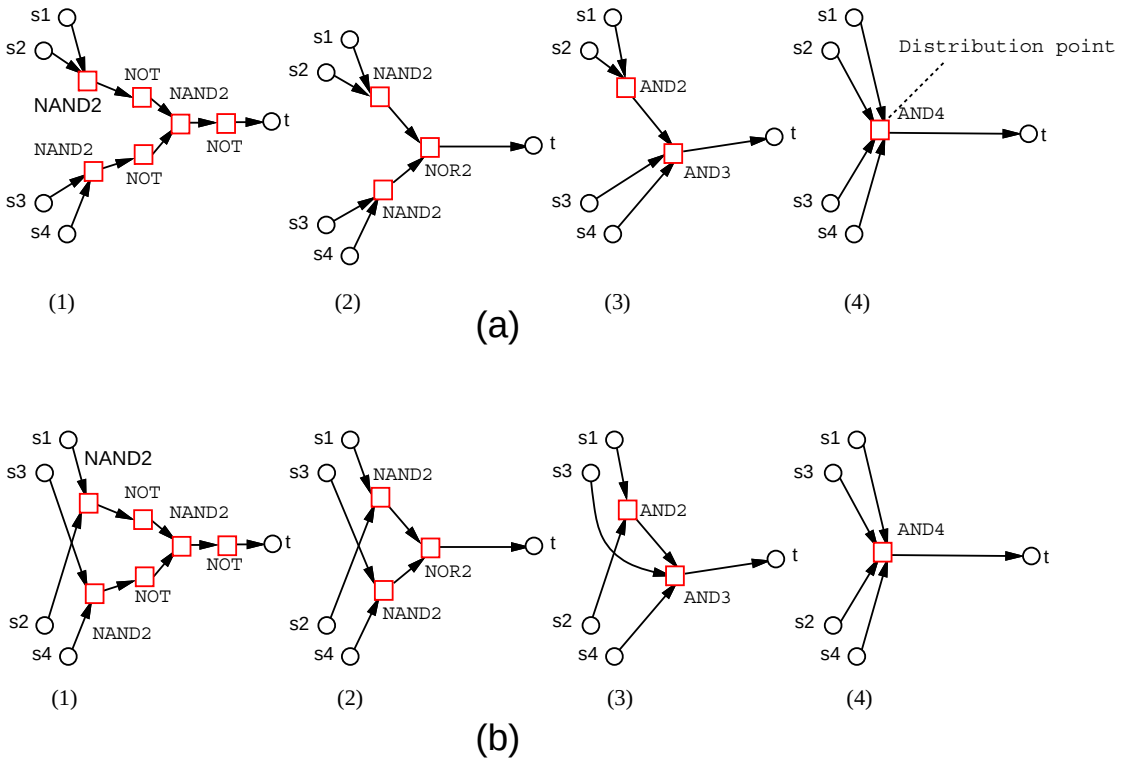
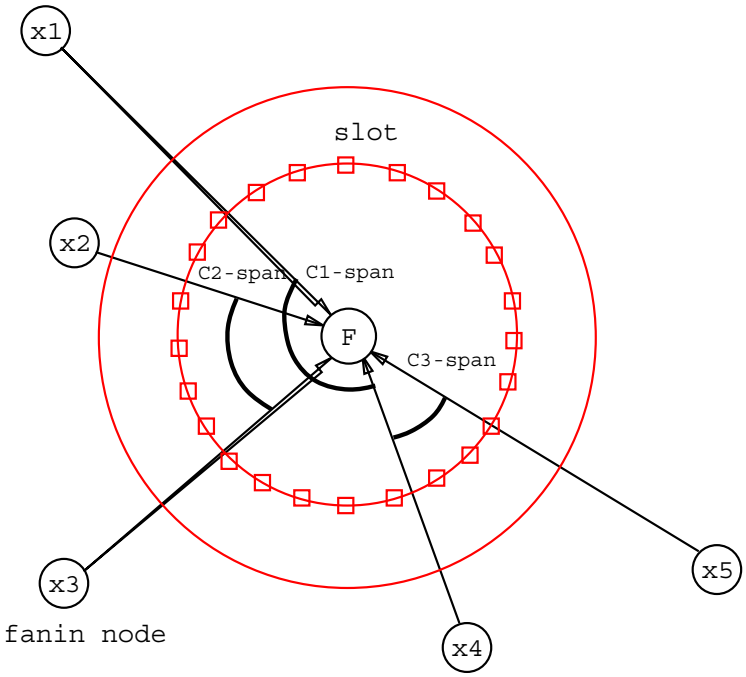


Figure 2: Active gate area versus wire length trade-off



F = X1 X3 X4 + X2 X3' + X4' X5

Figure 3: Cube ordering viewed as a linear assignment problem

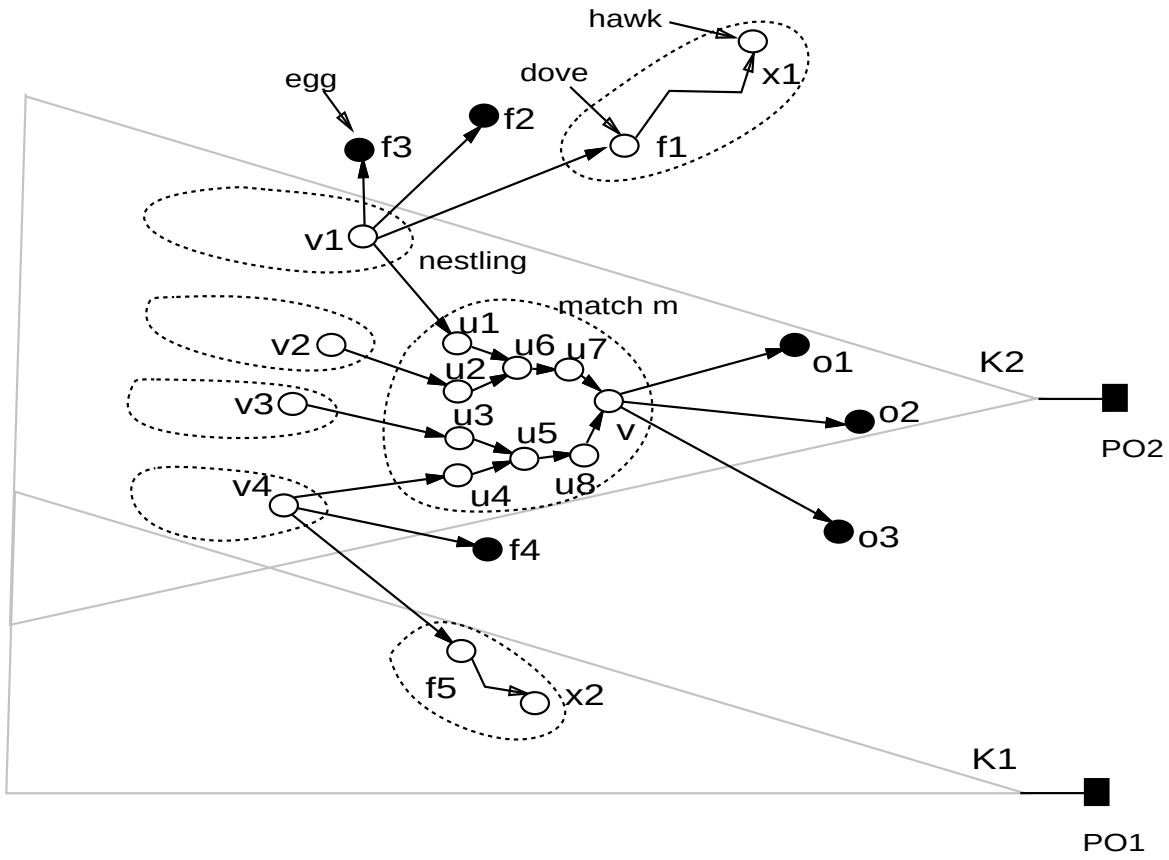


Figure 4: Cost calculation for a candidate match

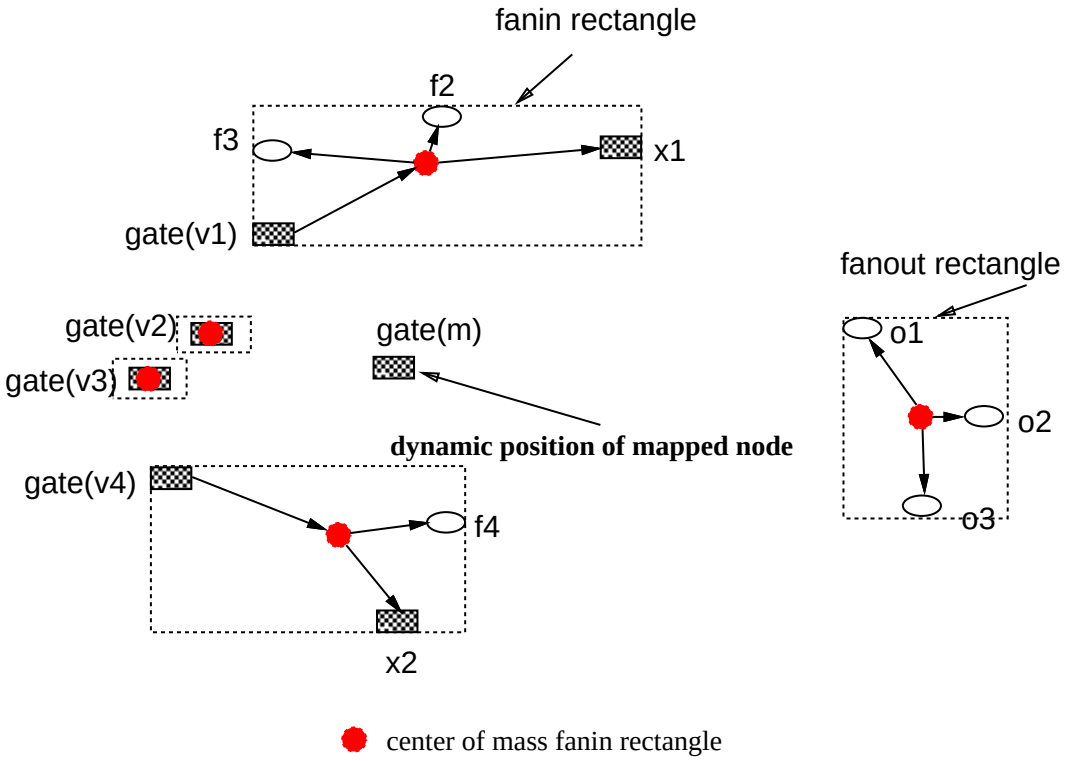


Figure 5: Dynamic updating of placement positions (Euclidean norm)

Ex.	original	optimized
9symml	277	183
C1355	1032	552
C1908	1497	535
C3540	2934	1283
C432	372	219
C5315	4369	1763
C880	703	414
apex6	904	732
apex7	289	243
b9	236	124
duke2	763	452
f51m	169	80
rot	764	664
z4ml	77	43

Table 1: Multi-level benchmarks: number of literals in factored form

example	MIS			LDTM		
	gate area mm^2	logic delay ns	cpu time s	gate area mm^2	logic delay ns	cpu time s
9symml	0.288	16.19	9.4	0.280	16.23	15.2
C1355	0.612	23.77	36.1	0.664	23.94	48.4
C1908	0.830	32.32	59.3	0.811	33.01	74.5
C3540	1.810	46.92	92.7	1.933	47.19	192.3
C432	0.374	28.09	24.9	0.400	27.56	38.5
C5315	2.300	37.10	75.6	2.305	40.35	181.5
C880	0.637	32.85	33.4	0.640	33.10	51.8
apex6	1.005	15.28	25.7	1.012	15.06	56.9
apex7	0.342	14.20	10.9	0.344	14.31	27.1
b9	0.187	5.84	5.2	0.189	6.25	8.8
duke2	0.655	16.28	30.3	0.670	17.63	51.5
f51m	0.120	22.15	3.9	0.116	22.37	5.7
rot	0.918	20.73	26.7	0.919	21.98	59.5
z4ml	0.080	10.11	3.1	0.087	10.21	4.5

Table 2: Comparison of the total gate area and logic delay after placement and routing. The cpu time reflects the mapping time.

example	MIS			LDTM		
	chip area mm^2	total delay ns	cpu time s	chip area mm^2	total delay ns	cpu time s
9symml	0.860	22.66	32.9	0.841	21.15	40.1
C1355	2.371	36.16	131.2	1.883	31.21	154.4
C1908	3.233	47.73	150.9	2.764	43.77	161.1
C3540	8.224	70.26	296.4	8.106	65.01	420.8
C432	1.197	37.46	67.2	1.101	34.84	82.1
C5315	10.754	60.85	776.1	9.522	55.66	861.0
C880	1.961	45.50	126.1	1.772	41.93	144.2
apex6	3.887	27.43	375.0	3.794	25.19	397.6
apex7	0.935	18.96	73.9	0.906	15.70	84.5
b9	0.491	8.03	42.1	0.486	7.52	45.8
duke2	2.604	25.25	100.9	2.556	23.19	120.1
f51m	0.303	29.34	17.1	0.267	27.67	18.6
rot	3.268	30.88	387.2	3.007	27.82	418.3
z4ml	0.179	12.72	13.0	0.170	11.90	14.6

Table 3: Comparison of the final chip area and circuit delay after placement and routing. The cpu time reflects the mapping, placement and routing times.