Technology Mapping Algorithms for Domino Logic

Min Zhao
Advanced Tools, Motorola Inc., Austin, Texas, 78729
and
Sachin S. Sapatnekar
Department of Electrical and Computer Engineering, University of Minnesota, 55455

In this paper, we present an efficient algorithm for technology mapping of domino logic to a parameterized library. The algorithm is optimal for mapping trees consisting of 2-input AND/OR nodes, and has a computation time that is polynomial in terms of constraint size. The mapping method is then extended to DAG covering that permits the implicit duplication of logic nodes. Our synthesis procedure maps the complementary logic cones independently when AND/OR logic is to be implemented, and together using dual-monotonic gates in the case of XOR/XNOR logic. The mapping procedure solves the output phase assignment problem as a preprocessing step. Based on a key observation that the output phase assignment could reduce the implementation cost due to the possible large cost difference between two polarities, a 0-1 integer linear programming formulation was formed to minimize the implementation cost. Our experimental results show the effectiveness of the proposed techniques.

Categories and Subject Descriptors: B.6.3 [**LOGIC DESIGN**]: Design Aids—*Automatic synthesis, Optimization*; B.7.1 [**INTEGRATED CIRCUITS**]: Types and Design Styles—*Advanced technologies, Standard cells, VLSI*; D.3.2 [**PROGRAMMING LANGUAGES**]: Language Classifications—*Specialized application languages*

General Terms: Algorithms, Design, Theory

Additional Key Words and Phrases: Domino logic, Technology mapping, Synthesis, Phase assignment, Parameterized library, Dual-monotonic gates, XOR/XNOR logic

1. INTRODUCTION

Domino logic is one of the most effective circuit configurations for implementing high speed logic designs. Although they are inherently non-inverting, and have the drawbacks of charge sharing and noise susceptibility, domino circuits offer the advantages of faster transitions and glitch-free operation. In the past, this logic family was applied only to the most timing critical paths, but the use of domino logic is now becoming much more widespread as designers begin to appreciate its

This research was supported in part by ?
A preliminary version of part of this paper appeared in the Proceedings of the IEEE/ACM International Conference on Computer-Aided Design, 1998 and the Proceedings of the IEEE International Symposium on Circuits and Systems, 2000.

advantages. With this growing degree of utilization, there is now a strong need for good design automation tools to support domino-based design.

A representative domino gate configuration is shown in Figure 1. It consists of an NMOS pull-down network, two clock controlled transistors (labeled A and B) and a static gate, usually an inverter, which is used to buffer between dynamic nodes. When the clock input is low, the gate precharges, charging the dynamic node d to logic 1. In the next half-cycle when the clock goes high, the domino gate evaluates, i.e., the dynamic node either discharges or retains the precharged state, depending on the values of the input signals.

The domino technology mapping problem is defined as: Given an optimized Boolean network and a set of permissible domino logic gates, implement the nodes in the network using those gates, such that the objective cost is minimized: this cost here may be, for example, the area, delay or power.

Technology mapping based on a static CMOS standard cell library is a well established problem originally addressed in [Keutzer 1987; Detjens et al. 1987; Touati et al. 1990]. In addition, various matching technologies are summarized in [Micheli 1994; Benini and Micheli 1997], and new decomposition, matching, cost calculation and gate selection procedures are proposed in [Touati et al. 1990; Chaudhary and Pedram 1995; Kukimoto et al. 1998; Jongeneel et al. 2000; Matsunaga 1998; Lehman et al. 1997]. Compared with static logic, the use of domino gates presents more choices to the designer. Moreover, their non-inverting property demands a more complex synthesis flow than for static logic. Therefore, domino circuits have conventionally been designed with a great deal of manual input from designers. Although most existing technology static mapping techniques can be migrated to domino logic easily, they may perform poorly if they fail to consider the special features of domino logic. For example,

—The absence of a complementary PMOS pullup network, lower short circuit currents and small driven loads at the dynamic node all motivate the use of large NMOS pull-down networks. In such a situation, it is difficult to implement the flexible functionality of domino gates with fixed cell libraries since the number of required cells is prohibitively large. Hence, a parameterized library is an appropriate option for domino gate mapping.

—DAG covering, which permits the overlap of two gates, is used in static logic mainly to reduce delay. For domino logic, it is important to reduce both delay and area. For circuits whose subject DAG composed of many small trees, DAG covering is observed to improve the performance of the domino mapper significantly over tree covering. Further details are furnished in Section 2.4.

—A number of configurations are available to domino logic that do not have exact equivalents for static logic. For example: the inverter at the output of the domino gate shown in Figure 1 can be replaced with other static gates [Williams 1996; Thorp et al. 1998]; XOR logic can be implemented with dual-monotonic gates [Williams 1996]; multiple output domino gates are used in high speed adder design [Wang et al. 1994; Wang et al. 1997]. Mapping to these gate types can be helpful to reduce the cost function.

—The non-inverting property of domino logic makes it important to reduce the logic duplication required to preserve unateness. Therefore, output phase assignment

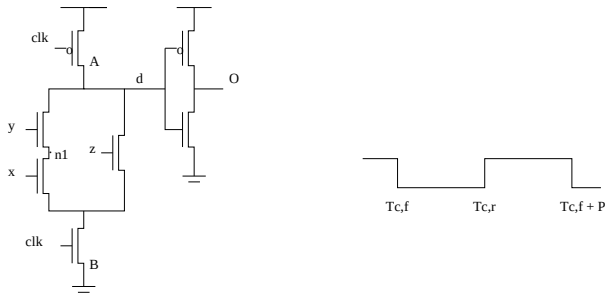

Fig. 1. A typical domino circuit

Fig. 1: `sapatne1.tex`

is performed as a preprocessing step before technology mapping to reduce the duplication, and therefore the area/power cost.

Recently, several papers considering these special properties have appeared. In [Prasad et al. 1997], a complex CMOS domino logic circuit synthesis flow for technology independent optimization and technology mapping of CMOS domino logic circuits was addressed. The work in [Puri et al. 1996] considers the problem of output phase assignment to minimize the duplication overhead required to make a network unate so that it may be implemented in domino logic. Other related work includes [Thorp et al. 1998; Zhao and Sapatnekar 1998; Puri 1998; Yee and Sechen 1997; Kim et al. 1999; Patra and Narayanan 1999].

The objective of this paper is to explore efficient technology mapping methods for domino logic that exploit the special features of domino style gates. A new approach for mapping circuits to a parameterized library is first proposed. Several methods that make use of the special features of domino logic are then developed and incorporated into the parameterized library mapper framework.

The paper is organized as follows. In Section 2, we present a novel parameterized library matching technique and extend the method to directed acyclic graph (DAG) covering. Section 3 suggests a technology mapping methodology that maps dual-rail logic cones independently for AND/OR logic, and together for XOR/XNOR logic. A method for incorporating the dual-monotonic domino gates into the framework of parameterized library mapping is also described. In Section 4, the reasons why output phase assignment influence the implementation cost are analyzed and the output phase assignment problem is formulated into a 0-1 programming problem. An existing linear programming package is applied to solve the problem as a preprocessing phase before technology mapping. The results for the technology mapping methods described in Sections 2 through 4 are presented in Section 5. Finally, we summarize our findings and present some concluding remarks in Section 6.

2. PARAMETERIZED LIBRARY MAPPING ALGORITHM

2.1 Problem definition

The concept of a parameterized library was originally proposed in [Detjens et al. 1987; Berkelaar and Jess 1988] for static complex gate mapping. A parameterized library is defined as a collection of gates that satisfy the constraints on the width (maximal number of parallel chains) and height (maximal number of series chains) of the pull-down or pull-up implementations of a gate. Such a target library for mapping the input network provides a great deal of flexibility and potential cost savings. Unlike a conventional library, the size of the parameterized library makes it infeasible to create cell layouts for all cells in the library. Therefore, the layout of cells in the library that are used in the mapped network is then produced by on-the-fly cell generation [Burns and Feldman 1998].

The application of parameterized library and on-the-fly cell generation to domino logic technology mapping was suggested in [Prasad et al. 1997], which further alluded to [Brayton et al. 1984; Ortiz and Lefebvre 1993]. This is particularly useful in the context of domino circuits since domino gates typically consist of a large NMOS pull-down logic network. The explanation for the use of large NMOS pull-downs is threefold. Firstly, the number of series transistors to the supply or ground

node in reasonable cells must be restricted, and the number of series transistors in the pulldown [pullup] network of a static CMOS gate is identical to the maximum number of parallel chains in the pullup [pulldown] network. In domino gates, in the absence of a PMOS counterpart, the number of parallel branches of the NMOS network may be large. Secondly, the NMOS network typically drives a small load, which permits the use of a slightly larger number of NMOS series transistors than for a static CMOS gate. Thirdly, each domino gate includes the overhead of two clock-controlled transistors, and therefore the less the number of domino gates, the less is the load that must be driven by the clock tree.

With this increase in the maximum number of series transistors and parallel chains in the NMOS network, the number of feasible gate functions in a library increases dramatically [Detjens et al. 1987]. Hence, using a fixed library greatly restricts the segment of the solution space that is searched during mapping, and instead, a parameterized library is more appropriate for technology mapping of domino logic.

In an environment that uses a parameterized library, we modify the definition of the domino technology mapping problem as: Given an optimized Boolean network and constraints on the width and height of the domino gates, implement the nodes in the network with domino logic gates, such that the objective cost is minimized.

2.2 Previous work and motivation

Our work is primarily related to two previously published methods for technology mapping to a parameterized library. The work in [Prasad et al. 1997] is a direct extension of the dynamic programming method used for static circuits that proceeds in a tree-by-tree manner [Detjens et al. 1987]. Matches are found by enumerating all possible domino gates at each node of the DAG. The drawback of such an approach is that at a particular node, under a given set of width and height constraints, the number of possible mapping schemes can be extremely large. Hence, recursively enumerating all possible solutions to obtain the optimal solution can be computationally expensive.

A second relevant piece of work is [Berkelaar and Jess 1988], where a top-down heuristic mapping method for a parameterized library is provided. The Boolean net is first decomposed into a multiple input AND/OR/NOT directed acyclic graph (DAG) network. At each mapping step, [Berkelaar and Jess 1988] limits its view of a certain expression to one level and maps as many nodes as possible at the current level into a complex gate. If the number of inputs at a node is large enough to exceed some specified size constraints, then a new gate is formed. Although the method provides the optimal solution for one level, the solution is greedy and is not guaranteed to be optimal over the entire network.

In this section, we will present a new parameterized library mapping algorithm that is optimal for a decomposed network in the form of a tree structure with two-input AND/OR nodes. The method has a computational complexity that is polynomial in terms of constraint size. One of the novel features of this approach is related to the way in which pattern matching is performed. In many cases (and particularly for a parameterized library), the matching of a node can relate to the matching of its children. To see this, consider the example shown in Figure 2, which shows a two-input AND/OR network. Suppose that cells a and b are matching

patterns for nodes E and F, respectively, and that there is a containment relation between cell a and b. To assert the existence of pattern b for node F, we need not traverse all the way to the leaf nodes of cell b. Instead, as long as we know that (i) cell a is a valid pattern of one child, (ii) cell c is a valid pattern of the other child, (iii) the library has the containment property described above, and (iv) the type of node F is AND, we can infer that cell b is a valid match for node F. This idea is somewhat similar to the Hoffman-O'Donnell approach [Hoffman and O'Donnell 1982]that has been used in [Jongeneel et al. 2000]

Therefore, from all the valid patterns of children, the containment relationships of the library, and the type of the current node, we can enumerate all of the possible complex gates that match the current nodes. In our parameterized library mapping method, we utilize this matching relation between parents and its children, combining the traditional bottom-up dynamic programming mapping approach with the constraint size equations provided in [Berkelaar and Jess 1988]

2.3 The algorithm

Given an arbitrarily optimized network, it is first unated [Puri et al. 1996] since domino circuits can only implement unate logic functions of the primary inputs. It is then mapped into a two input AND-OR DAG network, after which the DAG network is partitioned into two-input AND-OR trees. This is the starting point of our parameterized library tree-covering algorithm. The algorithm follows the traditional dynamic programming method [Cormen et al. 1990]. At each tree node, each stored subsolution is optimal for its subtree under specified constraints.

2.3.1 *Node data structure.* At each node, we store the optimal subsolutions for all possible [height,width] combinations from $[1, 1]$ to $[H, W]$. Therefore, there are a maximum of $H \times W$ optimal subsolutions that can be possibly stored for every node. Each optimal subsolution can be represented as $\{S, P, C, \{S_l, P_l\}, \{S_r, P_r\}\}$. Here, S $(1 \leq S \leq H)$ is the height constraint of the current node, P $(1 \leq P \leq W)$ is the width constraint of the current node and C is cost. Different combinations of the child node constraints can lead to the same parent node constraint, and of all these, $\{S_l, P_l\}$, $\{S_r, P_r\}$ is the combination that provides the minimal cost under the current constraints $\{S, P\}$.

Physically, $\{S, P\}$ represents a segment of a domino pull-down whose maximum height and width are S and P, respectively. The cost C is the accumulated cost of its fanin cones including the cost of the current domino circuit segment.

2.3.2 *Node constraint functions.* Here, we use the AND and OR formulas provided in [Berkelaar and Jess 1988] and list them here for convenience. The computation of the height and width of a subsolution at a parent node, formed by combining subsolutions at child nodes. Due to the series-parallel structure of the domino pull-down, we only need to consider two types of nodes: AND nodes and OR nodes. We assume the height and the width, respectively, of the subject node to be given by the pair $\{S, P\}$. The constraint on its left child is $\{S_l, P_l\}$, while that on its right child is $\{S_r, P_r\}$. The following represent all operations at a parent node:

(1) OR operation: $S = \max(S_l, S_r)$, $P = P_l + P_r$

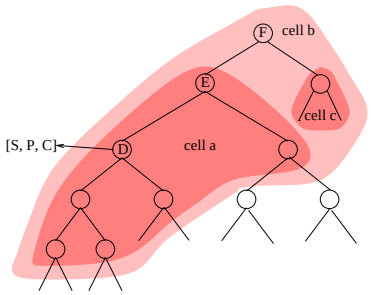

Fig. 2. Matching technique of parameterized library mapping

Fig. 2: `sapatne1.tex`

(2) AND operation: $S = S_l + S_r$, $P = \max(P_l, P_r)$
(3) gate formation operation: $S = 1$, $P = 1$

A gate formation operation corresponds to a situation where the structure collected so far (during the dynamic programming procedure) is converted to a domino gate with an output at that node.

2.3.3 *Node mapping algorithm*

```
ALGORITHM: Node Mapping
1.  for each valid [height,width] subsolution of the left child
2.      for each valid [height,width] subsolution of the right child
3.      {
4.        {S, P} = NODE CONSTRAINT FUNCTIONS ({S_l, P_l}, {S_r, P_r}) ;
5.        if {S, P} is within the constraints (H, W)
6.        {
7.          C = NODE COST FUNCTION (C_l, C_r)
8.          if (C < C[S, P]_min) then C[S, P]_min = C.
9.          if (C < C_min) then C_min = C.
10.       }
11.     }
12.  C[1, 1] = GATE FORMATION (C_min)
```

The $\{1, 1\}$ subsolution of a node is obtained from the subsolution $\{S, P\}$ of the same node whose implementation cost is minimal. A subsolution $\{S, P\}$ ($S > 1 \parallel P > 1$) at a parent node is obtained by combining optimal subsolutions at children nodes. The solution $C[1, 1]$ of the root of the decomposed tree is the optimal subsolution of the subject AND-OR tree.

2.3.4 *Node cost functions.* By defining different `NODE COST FUNCTION`'s for Line 7 of the algorithm Node Mapping, various cost models, such as area, delay, power and delay-area product can be applied to the parameterized library mapping algorithm.

Area cost model. Unlike the library-based technology mapping where the areas of cells are known beforehand, the cells of a parameterized-library are synthesized by the cell generators on-the-fly. Therefore, it should be possible to obtain an accurate area cost model by characterizing the cell generator and defining `NODE COST FUNCTION` from the characterization results. Otherwise, various other area cost models could be used, ranging from simple models that count the number of transistors or the number of gates, to more complex models that consider penalty factors for the clock routing cost, the extra shielding tracks, the precharge devices and keepers. For simplicity, in our explanation below, we adopt the number of transistors as the area cost. The `NODE COST FUNCTION` can then be defined as follows for each operation:

(1) leaf operation: if **is-leaf** $C = C + 1$.
(2) OR/AND operation: $C =$ leaf operation(C_l) + leaf operation(C_r)
(3) gate formation operation: $C = C_{min} + 4$

In Case 1, the status of **is-leaf** corresponds to a primary input or a situation where the pull-down structure accumulated so far was consolidated into a domino gate and therefore a new domino pull-down structure will be started at this point. In this situation, the input transistor of the next level of domino gates adds one more transistor to the cost; otherwise, the node is an internal node of the domino gate and no additional cost is required to be added.

In Case 2, C_l is the optimal subsolution for the left child under the constraint $\{S_l, P_l\}$ while C_r is the optimal subsolution for the right child under the constraint $\{S_r, P_r\}$. The number of transistors at an OR/AND operation is found by simply summing up the number in the subtrees, after taking the leaf operation into consideration. If the left child or the right child is the leaf node for a new domino gate, the area of this new domino gate segment will be included by conducting the leaf operation at children. (Not that if the node is not a leaf, the leaf operation leaves the cost unchanged.)

Finally, in Case 3, the gate formation operation involves the formation of a gate from the accumulated domino NMOS circuit segment. The minimal cost subsolution, C_{min}, from the set of subsolutions at that node is chosen. Above and beyond this cost, which measures the number of transistors in the domino pull-down network, we add the overhead of two clock control transistors and an inverter, corresponding to four more transistors. Therefore, we have $C = C_{min}+4$. We emphasize that the gate formation operation corresponds to the effect of creating a gate at the *current* node of the DAG, and the leaf operation considers its effect at the *next* node in the DAG.

These functions can be illustrated as shown in Figure 3. The AND operator places the transistors in series, and the corresponding $\{S, P\}$ values and cost are computed as shown. If, at this point, we were to decide to consolidate these into a gate, then the corresponding situation would correspond to four extra transistors as shown. The $\{S, P\}$ value of the last transistor would then be set to {1,1}.

In the above example, we have illustrated how to form the `NODE COST FUNCTION` for an area model that counts the number transistors. Other cost functions could also be incorporated into these operations. For example, if a keeper is always added to each domino gate and a domino gate is always "footed" (i.e., it has an NMOS clock controlled transistor), then additional clock tree routing costs are incurred, and the gate formation operation would have a cost of $C = C_{min}+4+C_{keeper}+C_{clk}$, where C_{keeper} and C_{clk} count the costs for the keeper transistors and the clock routing overhead.

Suppose there is a precharge device at each internal node of a domino gate, the AND operation should be defined as $C = \text{leaf operation}(C_l) + \text{leaf operation}(C_r) + C_{precharge}$ while OR operation is still same as before. The alteration to the cost for the AND operation stems from the fact that each AND operation produces a new internal node that is connected to VDD through a precharge transistor.

Delay model. As in the case of the area model, the delay model can be obtained by characterizing the library generator. Otherwise, the delay of a domino circuit may be measured using the number of levels of domino gates, or the Elmore delay model, or a look-up table, or the MIS library delay model [Detjens et al. 1987; Chaudhary and Pedram 1995]. Here, we will show how the pin-independent Elmore

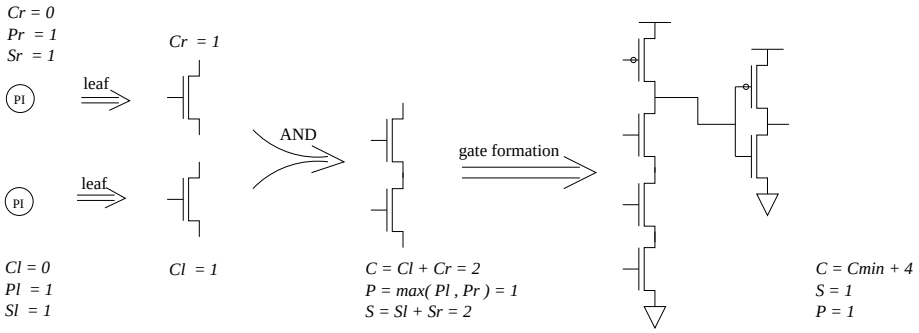

Fig. 3. An illustration of the area cost function

Fig. 3: `sapatne1.tex`

delay model is used. The delay models of library-based static mapping usually are complicated by the load-dependent property of the delay model and the difference between rising and falling delays. For simplicity, we assume here that the cells of parameterized library consist of uniformly sized transistors and leave the task of transistor sizing to a later step in the design flow of sizing optimization. Therefore, once we know the fanout of current node, we know the capacitive load for the delay model.

For a domino gate, the falling delay during the precharge stage will not be propagated to the next level of gates, and as long as the falling delay of each domino gate is smaller than its precharge duration, the timing constraint is satisfied. Therefore, falling delay usually is not in a critical path and is not considered during the technology mapping procedure, and only the rising delay of domino logic is considered in the delay model in our mapper. However, during the timing verification and optimization procedure after synthesis, the falling delay of each gate must be examined and reduced if the timing constraints are violated [Venkat et al. 1996; D. V. Campenhout et al. 1996; Zhao and Sapatnekar 2000].

The Elmore delay model similar to that in [Fishburn and Dunlop 1985; Sapatnekar and Kang 1993] has been used in this work, given by

$$D_{domino} = R_n \cdot (1 + S) \cdot (C_{dp} + k \cdot T \cdot C_{dn} + C_{gn} + C_{gp}) + R_p \cdot (C_{dp} + C_{dn} + fanout \cdot C_{gn}) \quad (1)$$

where R_n, R_p are the driving resistances, C_{gn}, C_{gp} the gate capacitances, and C_{dn}, C_{dp} the source/drain capacitances of the NMOS transistor and the PMOS transistor, respectively, S, P is the maximum height and width of the NMOS pull-down network, respectively and $fanout$ is the number of the fanout of the current node. T is the number of NMOS transistors in the NMOS pull-down network and $k(\leq 1)$ is a user-defined constant that reflects how the capacitance in the intermediate nodes influences the gate delay.

The first term in Equation (1) represents the delay of the NMOS pull-down network that drives the output inverter, where the clock-controlled transistor in series with the S transistors contributes the $(1 + S)$ factor. The second term represents the delay of the inverter that is driving the fanout gate capacitance. The driven capacitance here includes the source/drain capacitance of intermediate nodes of the driving gate, and the gate capacitances of the driven gates. Like the area model shown before, the delay model can be easily evaluated for the OR/AND operation and gate formation operation.

The procedure of Algorithm Node mapping with area minimization objective is illustrated in Figure 4. In this example, three-tuples are used to represent combinations of $\{S, P, C\}$. All primary inputs are initialized with tuple $\{1, 1, 0\}$. The tree is traversed from the leaf node upwards, and all subsolutions are enumerated using dynamic programming, eliminating any solutions that are suboptimal. From the dynamic programming viewpoint, the optimal solution under each constraint $\{S, P\}$ is a non-suboptimal substructure. The non-suboptimal subsolutions at a node are listed and the problem is solved by recursive enumerations of its higher level nodes.

For example, the AND operation on both $(\{2, 2, 3\}, \{2, 3, 5\})$ and on $(\{2, 1, 8\}, \{2, 3, 5\})$ produce $\{4, 3, C\}$. Only the tuple of this type with minimal cost and its correspond-

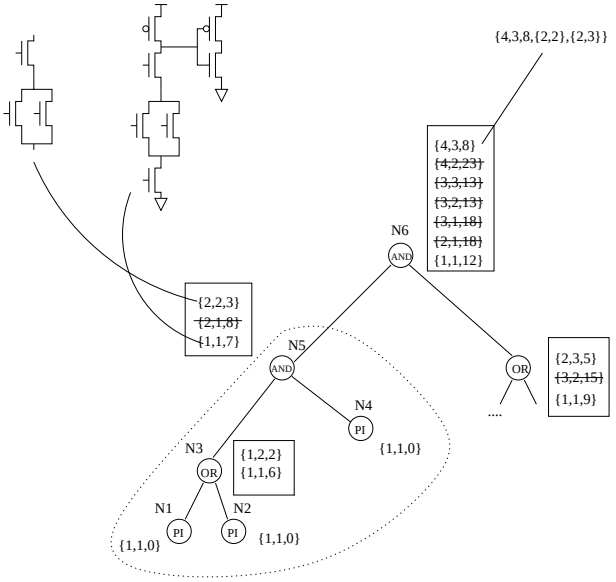

Fig. 4. An example illustrating the parameterized library mapping procedure

2.3.5 *An example.* Fig. 4: `sapatne1.tex`

ing child tuples are stored at node N6. This corresponds to $\{4, 3, 8\}$, which is a partial solution that may be used in the future. For the combination of $S = P = 1$, we choose the minimum cost solution at that node and construct a gate corresponding to that subsolution. For each child tuple that has a $\{1, 1, C\}$ configuration, the cost is incremented by 1 by the leaf operation. For example, $\{1, 1, 7\}$ is combined with $\{1, 1, 9\}$ at node N6 to give the tuple $\{2, 1, 18\}$. Finally, at node N6, the minimal cost obtained from all combinations of its children is 8, and the subsolution $\{1, 1, 12\}$ is obtained using the formula $C = C_{min} + 4$, corresponding to the gate formation operation.

Further pruning can be carried out by recognizing that several of these tuples are suboptimal. For instance, the tuple $\{3, 3, 13\}$ is suboptimal to the tuple $\{3, 2, 13\}$ since it has the same cost but with a lower value for both S and P. Carrying this argument further, it is easily verified that most of the tuples are suboptimal to $\{1, 1, 12\}$, so that the only tuples that are eventually maintained are $\{4, 3, 8\}$ and $\{1, 1, 12\}$.

2.3.6 *Complexity analysis.* From the above procedure, we can see that the space complexity of the algorithm is $O(WHN)$ and the time complexity is $O(W^2H^2N)$. where N is the number of tree nodes and W, H are the maximum allowable width and height of the domino gate, respectively. At each node, the AND-OR cost function will be executed at most $W^2H^2/2$ times, but generally the number of executions is much lower than this value.

2.4 DAG covering mapping

DAG covering uses a directed acyclic graph (DAG) as a subject graph for mapping and allows the overlap of two mapped cells. In this section, we will describe how to modify the parameterized library mapping technique proposed in the previous subsection to DAG covering.

2.4.1 *Motivations and previous work.* DAG covering is known to reduce the delay of the static circuits effectively [Kukimoto et al. 1998]. For domino logic, DAG covering is especially important as its influence on delay reduction is more significant. In addition, DAG covering may also help to reduce the area cost and reduce the clock load. The area cost of a domino gate is $K + 4$ while that of a static gate is $2K$, where K is the number of literals. If the transistor count of a tree is less than 4, then the area cost of the domino mapping will be larger than that of static mapping, even without considering duplication cost for unating [Prasad et al. 1997] In terms of delay, since the delay cost of a domino gate is the sum of the pull-down network delay and the inverter delay, a small NMOS evaluation network will introduce a large number of inverters in the path that could offset the advantage of fast switching speed provided by domino logic. Since large multi-level circuits often have a large number of multiple fanout nodes, the tree mapping based procedure that breaks the subject DAG at multiple fanout points often generates very small trees, leading to poor solutions in delay, area and loading on the clock tree. DAG covering provides one important method to overcome this problem.

Previous work on DAG covering of library-based static mapping includes [Keutzer 1987; Detjens et al. 1987; Chaudhary and Pedram 1995; Kukimoto et al. 1998], of which [Kukimoto et al. 1998] proposed a delay-optimal DAG covering algorithm

under load-independent delay model. In this section, we extend the parameterized library matching technique into DAG covering by applying a similar idea as [Kukimoto et al. 1998]. In our case, we use heuristics to remove the load-dependent delay assumption, and the additional delay and area cost caused by the overlapped logic are incorporated into the delay and cost model during the *correct cost estimation* step.

2.4.2 *Algorithm outline.* The conventional procedure of tree-by-tree mapping can be described as follows:

```
ALGORITHM: Tree-by-tree mapping
1.  Forward-traverse the network by topological order, at each node current
2.  {
3.     Node Mapping(current)
4.     if (fanout(current) > 1)
5.        Invalidate all the subsolutions except for [1,1]
6.  }
7.  Backward-traverse the network in topological order; at each node current
8.  {
9.     Assign best gate(current)
10.  }
```

In the above procedure, all subsolutions except for $[1, 1]$ are removed at the fanout node. DAG covering algorithm differs from tree-by-tree mapping in two respects. Firstly, instead of invalidating all subsolutions except for $[1, 1]$ on Line 6 of Algorithm *Tree-by-tree mapping*, we insert a step that correctly estimates the cost of each subsolution. In other words, all subsolutions of current node are kept and their cost estimates are adjusted to include the additional duplication cost from the overlaps between cells. Secondly, due to the possible existence of contradictions from multiple fanout nodes, *Assign best gates* on Line 9 becomes more complex for DAG mapping.

2.4.3 *Correct cost estimation.* During the DAG mapping procedure, all of the subsolutions are maintained at the fanout node. An additional cost in both delay and area will be added to the cost of the subsolutions, as shown in Figure 5.

The shaded area would be duplicated $fanout$ times, and therefore the additional area cost is $(fanout - 1) \cdot dupcost$, where *dupcost* represents the duplication cost. At the inputs of the duplication region, the number of the fanouts will increase and therefore the additional delay cost of the gate in the previous level is $R_p \cdot (fanout - 1) \cdot C_{gn}$. As in [Chaudhary and Pedram 1995], we divide the area contribution of each node by the fanout count of the node.

2.4.4 *Assign best gate.* The successors of a node determine its candidate solutions during the backward traversal. For a node that is connected to multiple fanouts, the solutions that are optimal for each fanout tree may be inconsistent, i.e., there may be a contradiction between these subsolutions, with different mapping solutions being optimal for different outputs, as well as between the delay-area points if the delay-area points if the delay-area tradeoff curve [Chaudhary and Pedram 1995] is

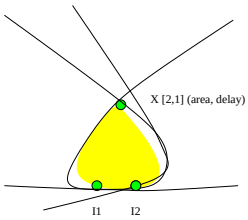

Fig. 5. Correct cost estimation

Fig. 5: `sapatne1.tex`

used. In the case of area optimization, we choose the subsolution with the minimal height and width out of all candidates to minimize the area cost of duplication. If delay optimization is the objective, then we choose the solution that satisfies the delay requirement.

3. DUAL-MONOTONIC GATE MAPPING

3.1 Dual-monotonic logic

A dual-monotonic gate is a merged gate that generates both negative and positive polarities of a logic signal [Williams 1996]. A typical dual-monotonic two-input XOR gate is shown in Figure 6. The logic used to implement $a \cdot b$ and $\bar{a} \cdot b$ share the same transistor, D. Due to the complementary relations between a and $\bar{a}$, a sneak path between $O1$ and $O2$ can be prevented. Moreover, a dual-monotonic gate is not limited to be just an XOR gate. More flexible configuration of dual-monotonic gates can be formed by replacing b and $\bar{b}$ with another XOR gate or the other logic.

While dual-monotonic implementations of XOR logic use a smaller number of transistors than duplicated gates, this is not true for all logic functions. Indeed, implementations of most common dual-monotonic gates do not share as many transistors as the XOR gate.

3.2 Previous work and motivation

A domino gate implementation of an input network often requires the synthesis of both positive and negative signals at many nodes due to the unateness requirements; this is known as dual-rail logic. A logic function and its complement can be built as two separate gates or as one merged gate (a dual-monotonic gate).

If we examine several papers on domino logic synthesis that have appeared [Prasad et al. 1997; Thorp et al. 1998; Zhao and Sapatnekar 1998], we see that all of them follow the basic synthesis methodology proposed in [Prasad et al. 1997] in which dual-rail logic cones are independently mapped. This approach has the advantage of maintaining the mapping flexibility of each polarity. However, the presence of an XOR function and its reconvergence property will decompose the input network into very small mapping trees, which causes a large area and delay cost for tree-by-tree technology mapping. On the other hand, dual-monotonic XOR is a widely-used configuration in manually designed domino networks; its application to synthesis of domino logic can effectively solve the above problem. In contrast, a three-input XOR dual-monotonic gate has to be implemented with four individual domino gates if the tree mapping method is used. Therefore, we propose a mapping method that can make use of the advantages of both cases by mapping dual-rail AND/OR logic independently with standard domino gates, and mapping XOR/XNOR logic with merged dual-monotonic gates.

3.3 Dual-monotonic mapping algorithm

After the unating stage where the inverters are removed, a signal and its complement become two separate signals and the complementary relationships between the signals are lost. Hence, the existing mapping techniques cannot be applied directly.

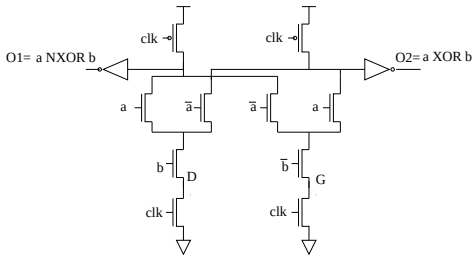

Fig. 6. An example of a dual-monotonic gate

Fig. 6: `sapatne1.tex`

One method to enable the mapping of dual-monotonic gates is to add a special type edge between complementary signals, named the *inverting edge*. In this case, the graphs representing subject networks or cells consist of the normal edges and the *inverting edges*, and dual-monotonic gates can be mapped by checking the isomorphism of these graphs. However, these *inverting edges* will significantly increase the complexity of the subject and cell graphs, and how to effectively map the flexible configurations of dual-monotonic gates becomes a difficult issue to solve.

Here, a heuristic method that collapses XOR/XNOR logic before the unating stage is used. The method consists of the following steps:

Step 1. Recognize the XOR/XNOR logic inside the DAG using a graph isomorphism based method. If both the positive and the negative polarity of XOR/XNOR logic are required, the XOR/XNOR logic networks are collapsed into one XOR/XNOR node to be implemented as a dual monotonic gate.

Step 2. Unate the DAG network and generate a network composed of AND/OR/XOR/XNOR nodes.

Step 3. Perform the technology mapping on the AND/OR/XOR/ XNOR subject network, mapping AND/OR nodes to the standard domino gates and XOR/XNOR nodes and its surrounding nodes to various dual-monotonic gates.

The last step of the dual-monotonic mapping algorithm is technology mapping on an AND/OR/XOR/XNOR network. During technology mapping, all possible matchings are enumerated at each network node. While the traditional mapping patterns can be applied to AND/OR nodes, the matching patterns available to XOR/XNOR nodes need to be considered. One single XOR/XNOR node can be mapped to an XOR dual-monotonic gate. Moreover, the flexible configurations of dual-monotonic gates enable the exploration of more matching patterns at XOR/XNOR nodes. The mapped gates could be gates other than XOR/XNOR gates as long as sneak paths are prevented: in Figure 6, this is achieved since a and $\bar{a}$ are logic complements. A more complex example is the matching pattern of Figure 7(a), which corresponds to a three input XOR gate as Figure 7(b).

Another example is the matching pattern of Figure 8(a), which corresponds to the dual-monotonic gate as Figure 8(b). It is obtained by replacing the transistors D and G in Figure 2 by NMOS subnetworks.

In this work, we incorporate the above issues in the parameterized mapper described earlier in this paper, but any other mapper may easily be adapted for this purpose with this heuristic method.

4. OUTPUT PHASE ASSIGNMENT USING 0-1 ILP

4.1 Previous work and motivation

The polarity assignments at the output of a domino circuit significantly influence the implementation cost. The output phase assignment problem is defined as follows: Given a combinational logic network and all primary inputs in the true and complemented form, choose an optimal phase (i.e., polarity) assignment for the primary outputs so as to implementation cost was minimized.

The output phase assignment problem was first addressed and solved in [Puri et al. 1996]. Domino logic is inherently non-inverting and removal of the intermedi-

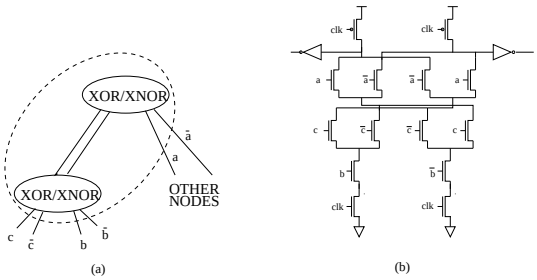

Fig. 7. Matching pattern 1: (a) the subject graph (b) its mapping to a three input XOR gate

Fig. 7: `sapatne1.tex`

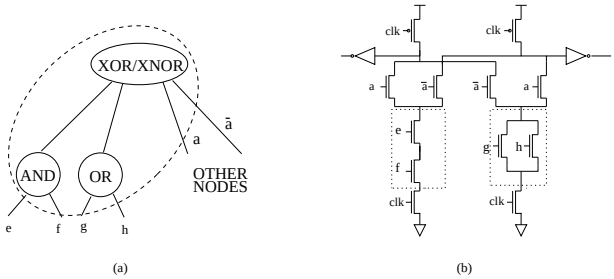

Fig. 8. Matching pattern 2: (a) the subject graph showing arbitrary AND/OR/XOR logic (b) its mapping to a dual monotonic gate

Fig. 8: `sapatne1.tex`

ate inverters requires logic duplication for generating both the negative and positive signal phases, which results in significant area overhead. In [Puri et al. 1996], it was found that this duplication area overhead can be substantially reduced by selecting an optimal output phase assignment. In [Puri et al. 1996], given an input network, the network is divided into the region that must be duplicated and region whose duplication cost can be minimized by output phase assignment, referred to as the optimizable logic region. The fanout nets in the optimizable logic region are called candidate nets. The problem of finding optimal output phase assignment to minimize the duplication of optimized logic region was formulated as 2SAT unate covering problem and solved by using binary decision diagrams.

In this paper, we consider the output phase assignment as a preprocessing step before technology mapping of domino logic. Our solution to the output phase assignment problem improves upon [Puri et al. 1996] in the following aspects.

—In our mapper, the duplication cost reduction problem is formulated into a 0-1 integer programming problem and a standard linear programming package can be used to solve the problem.

—We make a new observation that the output phase assignment accomplishes the objective of reducing implementation cost through another important factor: the cost difference between the implementations of positive and negative polarity. This polarity cost difference problem is also formulated into an 0-1 integer programming formulation. Combining this with the linear constraints for duplication reduction, the optimization problem of output phase assignment to minimize the implementation cost is solved.

The cost difference between the implementations of positive and negative polarity is caused from the fact that the logic evaluator of a domino gate composed of only NMOS. Suppose W and H are the constraints on the width (maximal number of parallel chains) and height (maximal number of series chains) of the NMOS pull-down network of domino gate, respectively. Due to the absence of a complementary PMOS network in domino gate, domino gates usually have large W with limited H, which can lead to the cost difference between implementations of positive and negative polarity. For example, given the constraints that $H = 2$ and $W = 4$, the logic network of Fig 9(a) will be mapped to three domino gates while its complement in Fig 9(b) requires only one domino gate. Moreover, the area overhead in addition to the NMOS logic evaluator for each domino gate is large. In the above example, the implementation cost of the logic network in Fig 9(a) is 18 transistors while its complement requires only 7 transistors. The area and delay cost of the latter is better than that of the former.

The unated DAG network consist of two parts: the part for which both polarities must be implemented, called the duplicated part, and the part for which only one polarity need be implemented. On the one hand, the phase assignment of the output nodes decides which segment of DAG network is to be duplicated; on the other hand, the output phase assignment decides which polarity is to be implemented for the non-duplicated part. Both factors have an important influence on the implementation cost. A given polarity assignment for the set of output nodes can influence these two factors in opposite ways in terms of on cost reduction, so that it is important to find the optimal solution that considers both factors simul-

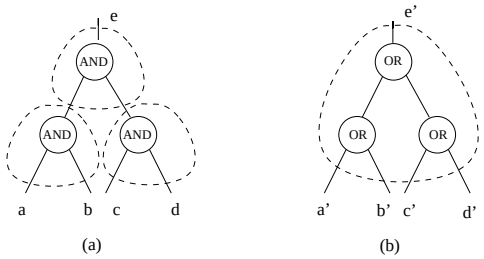

Fig. 9. The cost difference of (a) positive and (b) negative polarity

Fig. 9: `sapatne1.tex`

taneously. In the following discussion, a 0-1 programming formulation is provided for this purpose.

4.2 Algorithm outline

In our domino synthesis flow, output phase assignment optimization is a preprocessing step before technology mapping and is performed before unating the input network. After output phase assignment optimization, the inverters are added to the output nodes whose phases are assigned to be negative and then pushed towards the inputs network during the unating procedure. The output phase assignment optimization algorithm can be outlined as follows:

(1) Decompose the network into a disjoint set of trees and obtain a fast area cost estimation of each polarity of each tree. This can be obtained by counting the number of nodes in the tree or performing a fast tree-by-tree mapping on the network. The area cost of a tree is assigned to the root node of the tree, u, as the weight $C(u)$.
(2) Find the optimizable logic region using the method of [Puri et al. 1996]; build a DAG from the multiple fanout nodes of the optimizable logic region.
(3) Write out the 0-1 integer linear programming formulation for duplication cost minimization described in Sections 4.3 and for polarity cost difference described in 4.4, and then solve the combined formulation.

4.3 0-1 ILP for duplication cost minimization

This section describes the 0-1 formulation to minimize the duplication cost. Given an input network, a DAG $G(V, E)$ can be built as follows. Each vertex $v \in V$ corresponds to a multiple fanout or a primary output node in the optimizable logic region of input network. If vertex u is a literal node of a tree rooted at vertex v in the original input network, there is a corresponding edge $e_{u,v} \in E$ in DAG.

In the DAG, several constants are assigned to an edge or a node from the original input network. These include:

. $O(u, v)$, which represents the inversion polarity between vertex u and vertex v. If there is an even number of inverters from vertex u to vertex v in the input network, $O(u, v)$ is 0; otherwise, $O(u, v)$ is 1.

. $C(u)$, which represents the area cost of the tree whose root is at vertex u in the input network.

. $k(u)$, a constant whose value is twice the number of fanout of vertex u.

The $\{0, 1\}$ integer variables that will be included in the 0-1 programming formulation include:

. $r(u) = 1$ if there is an inverter moving from the inputs of node u towards the outputs of node u; else 0.

. $x(u, v) = 1$ if there is an inverter on the edge $e_{u,v}$ after the output phase assignment; else 0.

. $y(u, v)$ is a dummy variable that transforms a condition statement into a linear constraint.

. $q(u) = 1$ if the fan-in tree of node i needs to be duplicated; else 0.

The weight of an edge $e_{u,v}$ after output phase assignment, denoted by $w(u,v)$ is given by

$$w(u,v) = O(u,v) + r(u) - r(v) \tag{2}$$

Since $O(u,v)$, $r(u)$, $r(v) \in \{0,1\}$, $w(u,v)$ takes a value in the set $\{-1,0,1,2\}$. If $w(u,v) = 0$ or 2, it represents the absence of an inverter between node u and node v. If $w(u,v) = -1$ or 1, then there is one inverter between node u and node v. Therefore, this may be captured by the condition

$$x(u,v) = \begin{cases} 0 & w(u,v) \in \{0,2\} \\ 1 & w(u,v) \in \{-1,1\} \end{cases} \tag{3}$$

The above condition may be rewritten as a linear equation.

$$w(u,v) + x(u,v) = 2 \times y(u,v) \tag{4}$$

where $y(u,v) \in \{0,1\}$ is a dummy variable introduced to transform a condition statement into a linear constraint.

Combining the equations (2) and (4), we have

$$2 \times y(u,v) - x(u,v) = O(u,v) + r(u) - r(v) \tag{5}$$

If there is an inverter at any position in the fanout cone of a node u, the node will have to be duplicated to ensure the unateness property for domino logic. This condition can be given by the linear constraint

$$q(u) \times k(u) - \sum_{i \in dir-succ(u)} (x(u,i) + q(i)) \geq 0 \tag{6}$$

It can be illustrated as Figure 10. Here, $i \in dir - succ(u)$ implies that there is an edge from node u to node i in the DAG $G(V,E)$ defined in Section 4.3. The constant $k(i)$ was defined earlier and can easily be verified to always be larger than $\sum_{i \in dir-suc(u)(u)}(x(u,i) + q(i))$. This constraint implies that if any of its successors needs to be duplicated, or if there is one inverter at the output edges of node u, node u will have to be duplicated as $q(u)$ is forced to be 1; otherwise the objective function will force $q(u)$ to 0.

Therefore, the output phase assignment problem can be formulated as the 0-1 integer linear programming problem:

$$\begin{gathered} \text{minimize } \textstyle\sum_{u \in V} C(u) \times q(u) \\ \text{subject to} \\ 2 \times y(u,v) - x(u,v) = O(u,v) + r(u) - r(v) \quad \forall e_{u,v} \in E \\ q(u) \times k(u) - \textstyle\sum_{i \in dir-succ(u)}(x(u,i) + q(i)) \geq 0 \quad \forall u \in V \\ q(u), r(u) \in \{0,1\} \quad \forall u \in V \\ y(u,v), x(u,v) \in \{0,1\} \quad \forall e_{u,v} \in E \end{gathered}$$

4.4 0-1 ILP for polarity cost difference

This section describes the 0-1 formulation to minimize the implementation cost by considering the cost difference between positive and negative polarity implementa-

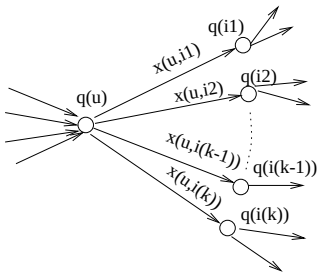

Fig. 10. 0-1 programming constraints

Fig. 10: `sapatne1.tex`

tions. In addition to the notation defined in Section 4.3, the symbols that will be used include the following:

$Z(u)$ is a constant that represents the cost difference between optimal implementation of each of the two polarities. If the implementation cost of positive[negative] polarity of the node u is $p(u)[n(u)]$, then the value of $Z(u) = p(u) - n(u)$.

We introduce $t(u)$ as a variable that helps to maximize $r(u)$. Its value can be expressed as follows:

$$t(u) = \begin{cases} 0 & q(u) = 1 \\ r(u) & q(u) = 0 \end{cases} \tag{7}$$

Similarly, we also define a variable $s(u)$to minimize $r(u)$ as

$$s(u) = \begin{cases} 0 & q(u) = 1 \\ 1 - r(u) & q(u) = 0 \end{cases} \tag{8}$$

There are three possibilities for each node after output phase assignment. A node will either be duplicated, implemented in positive polarity, or in negative polarity. In the case of no duplication at node u, if $Z(u) > 0$, it costs less to implement the node with negative polarity; if $Z(u) < 0$, it is better to implement the node u with positive polarity. Hence, using the previous definitions of $q(u)$ and $r(u)$, the problem can be expressed as

$$\begin{aligned} &\text{if } q(u) = 0 \textbf{ and } Z(u) > 0, \text{maximize } r(u); \\ &\text{if } q(u) = 0 \textbf{ and } Z(u) < 0, \text{minimize } r(u). \end{aligned}$$

The first statement can be expressed by linear equations

$$\begin{aligned} \text{minimize} \quad & -Z(u) \times t(u) \\ \text{subject to} \quad & t(u) \le 1 - q(u) \\ & t(u) \le r(u) \\ & q(u), r(u), t(u) \in \{0, 1\}, \qquad \text{if } Z(u) > 0, \end{aligned}$$

When $q(u) = 1$, $t(u)$ is forced to 0. Hence, there is no limit on $r(u)$ and the duplication cost is given by $C(u) \times q(u)$. When $q(u) = 0$, $r(u)$ is forced to be as large as possible since the negative value of $-Z(u)$. When polarity preferences of two nodes conflict with each other, $Z(i)$ is the weight that decides which node should win. Similarly, the second statement can be captured by

$$\begin{aligned} \text{minimize} \quad & Z(u) \times s(u) \\ \text{subject to} \quad & s(u) \le 1 - q(u) \\ & s(u) \le 1 - r(u) \\ & q(u), r(u), s(u) \in \{0, 1\}, \qquad \text{if } Z(u) < 0 \end{aligned}$$

4.5 0-1 ILP for minimal implementation cost

Combining with the linear constraints of Section 4.3 and Section 4.4, the 0-1 ILP formulation of the output phase assignment problem for cost minimization can be rewritten as

$$\text{minimize} \sum_{u \in V} C(u) \times q(u) - \sum_{Z(u)>0} Z(u) \times t(u) + \sum_{Z(u)<0} Z(u) \times s(u)$$

subject to

$$\begin{aligned}
2 \times y(u,v) - x(u,v) &= O(u,v) + r(u) - r(v) \quad \forall e_{u,v} \in E \\
q(u) \times k(u) - \sum_{i \in dir-succ(u)} (x(u,i) + q(i)) &\geq 0 \quad \forall u \in V \\
t(u) \leq 1 - q(u), \quad t(u) \leq r(u) &\quad \forall Z(u) > 0 \\
s(u) \leq 1 - q(u), \quad s(u) \leq 1 - r(u) &\quad \forall Z(u) < 0 \\
q(u), r(u), t(u), s(u) \in \{0,1\} &\quad \forall u \in V \\
y(u,v), x(u,v) \in \{0,1\} &\quad \forall e_{u,v} \in E
\end{aligned}$$

5. EXPERIMENTAL RESULTS

Our technology mapping package has been implemented using C++. The parameterized library mapping procedure of Section 2 constitutes the basic frame of the mapper, and the other methods are incorporated. The experiments were executed on the LGSynth91 multi-level combinational circuit sets.

All of the input circuits are first optimized with *script.rugged* of SIS. The input circuits for technology mapping of domino logic are unate equivalents of the benchmark circuits. They are obtained from the benchmark descriptions by first being optimized, then pushing the inverters as close to the inputs as possible, and finally duplicating the fanin cones of the inverters. Except for the results in Table IV and Table V, all of our results were obtained under the constraints of $W = H = 4$.

Our technology mapping procedure provide mapping with objective of
(1) area(delay) minimization,
(2) area minimization under the delay constraint.
The delay model described in Section 2.3.4 is applied for the delay estimation and area is estimated as the transistor count. Here, the transistor count is just coarse measurement of the area. The other factors that impact the area such as transistor type(PMOS or NMOS), the source/drain sharing, the clock routing cost, and additional noise reduction configurations are ignored.

In this section, our experimental results show the comparison of our work with previous works, the effectiveness of various enhancement methods discussed in Sections 2 through 4, as well as a comparison of domino implementations with CMOS static logic.

5.1 Effectiveness of the parameterized library algorithm

5.1.1 *Tree-by-tree mapping compared with [Berkelaar and Jess 1988].* Table 5.1.1 shows the effectiveness of our algorithm by comparing our results with those of [Berkelaar and Jess 1988]. Column 2-3, 4-6 shows the results of the delay-minimization objective, area-minimization objective with our tree-by-tree parameterized library mapping algorithm, respectively. For each set of results, delay, the number of

Table I. Effectiveness of the parameterized library mapping algorithm

Circuits	Our delay-minimization		Our area-minimization			[23]		
	Delay	#T/#G	#T/#G	Delay	CPU	#T/#G	Delay	CPU
b9	0.19	296/40	261/33	0.21	0.1	291/39	0.21	0.1
c8	0.25	307/38	282/633	0.27	0.1	347/46	0.27	0.1
count	0.56	357/46	357/46	0.58	0.1	362/47	0.58	0.1
i6	0.62	763/76	763/76	0.62	0.1	768/77	0.62	0.1
C880	0.83	1168/139	1163/136	0.90	0.1	1288/161	0.90	0.1
C1355	0.44	1844/232	1824/228	0.46	0.1	1844/232	0.46	0.1
C1908	0.75	2003/269	1978/264	0.80	0.1	2063/281	0.80	0.1
C2670	0.65	2047/225	1992/214	0.76	0.1	2477/311	0.76	0.1
C3540	1.03	4587/550	4527/538	1.07	0.2	5102/653	1.07	0.2
C6288	3.12	14172/1897	13702/1803	3.12	0.9	13767/1816	3.12	0.8
C7552	1.04	7989/965	7924/952	1.08	0.6	8784/1124	1.08	0.5
t481	0.61	1752/228	1697/217	0.72	0.1	1897/257	0.72	0.1
rot	0.50	1807/224	1777/218	0.53	0.1	1997/262	0.53	0.1
dalu	0.91	2375/276	2360/273	0.95	0.1	2755/352	0.95	0.1
k2	0.69	2914/417	2884/411	0.69	0.1	2864/407	0.69	0.1
des	1.83	10505/1249	9945/1137	1.83	0.8	11465/1441	1.83	0.8

transistor and number of gates are reported. In Column 6, the CPU time of our area-minimization mapping is listed. In Column 7-9, The results of the algorithm from [Berkelaar and Jess 1988] provided by our implementation are shown in terms of area, delay and CPU time. Here, the same delay model as Section 2.3.4 are used for [Berkelaar and Jess 1988] algorithm.

From Table 5.1.1, we can see that our method functions at approximately the same speed as the method in [Berkelaar and Jess 1988], but we show better results in terms of both area and delay. This is because that [Berkelaar and Jess 1988] is a greedy method, while our method considers all of the valid patterns and picks up the best. Moreover, the objective of [Berkelaar and Jess 1988] is to merge as many nodes as possible into gates, and no objective model can be incorporated very easily into the algorithm. Our algorithm provides the flexibility of incorporating various cost models, where significant differences in the solution are shown when the optimization objective is changed. The results in Table 5.1.1 are obtained using the tree-by-tree mapping method. As stated earlier, we find that the tree covering method can partition the DAG into small fanout-free regions, and the search space for the problem is very restricted. With the use of DAG mapping, which allows a larger search space, we expect further improvement of our results over [Berkelaar and Jess 1988]. However, since [Berkelaar and Jess 1988] do not present their results on DAG mapping, such a comparison is not easily possible.

One significant exception is the benchmark circuit $k2$, where our solution is worse than [Berkelaar and Jess 1988]. This can be attributed to the fact that it contains many multiple-input AND(OR) nodes whose fan-ins are larger than the constraint size $W(H)$ in the original network. Although our parameterized library algorithm is optimal for a 2-input network, the procedure of decomposition of the AND(OR) tree into a 2-input AND(OR) tree is not optimal, which can be illustrated on an example network in Figure 11.

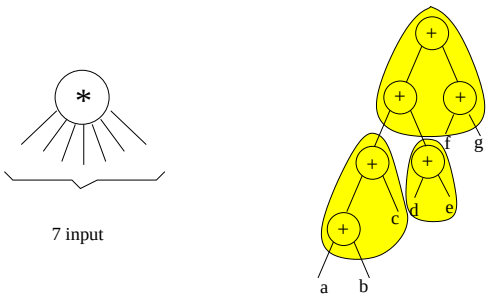

Fig. 11. The influence of decompose to the parameterized library mapping

Fig. 11: `sapatne1.tex`

Given the specification of a maximum of 4 parallel branches, our parameterized library procedure would require 3 gates for the given decomposition, while [Berkelaar and Jess 1988] can bind the network with two gates since it works on the multiple-input network directly. To overcome this problem, we would suggest that the multiple-input gate be decomposed in accordance with the constraints, i.e., first decompose the multiple-input AND(OR) node into OR nodes with a maximum of H inputs and AND nodes with a maximum of W inputs, and then decompose the W(H) input nodes into two-input nodes. With this decomposition method, the result of result of $k2$ can be improved to "2829/16." However, this decomposition method is only effective to a circuits having a large number of n-input OR(AND) nodes, where $n >> W$ or $n >> L$. We observe that this situation is not common for LGSynth91 benchmarks and has very little effect on the other benchmarks.

The main advantage of our algorithm is its fast speed while it gives the optimal solution for tree-by-tree mapping as the other library-base methods. The computation time of other domino mappers that directly match all the patterns by enumeration, such as [Prasad et al. 1997], is not available. However, from the CPU time of a static mapper, SIS, which will be shown at Table VI, and the complexity analysis in Section 2.3.6, we believe that the computation time required by our domino mapper should be much smaller.

Table II. DAG covering mapping

Circuits	DAG delay-minimization			DAG area-minimization			CPU
	Delay	#T/#G	Δ Delay	#T/#G	Delay	Δ Area.	(s)
b9	0.15	255/29	11.8%	243/27	0.18	6.9%	0.1
c8	0.17	298/35	26.1%	274/31	0.21	2.8%	0.1
count	0.36	327/37	35.7%	321/36	0.36	10.1%	0.1
i6	0.60	1065/109	3.2%	763/76	0.62	0.0%	0.1
C880	0.71	1139/124	12.4%	1127/123	0.74	3.1%	0.3
C1355	0.38	1768/152	11.6 %	1576/124	0.41	13.6%	0.4
C1908	0.57	1912/201	24.0 %	1693/166	0.66	14.4%	0.7
C2670	0.51	1993/191	12.1%	1912/179	0.69	4.0%	0.6
C3540	0.77	4374/466	21.4%	4209/441	0.95	7.0%	3.5
C6288	2.17	11935/1050	25.9 %	11578/974	2.86	15.5%	20.5
C7552	0.86	7503/761	13.1%	7373/741	1.05	6.9%	5.2
t481	0.46	1694/201	13.2%	1611/188	0.62	5.1%	0.3
rot	0.42	1746/199	12.5%	1673/186	0.52	5.9%	0.3
dalu	0.74	2286/239	14.9%	2239/233	0.90	5.1%	1.1
k2	0.50	2688/324	23.1 %	2548/302	0.65	11.7%	0.4
des	1.75	10261/1092	3.3%	9493/981	1.79	4.5 %	3.5

5.1.2 *DAG mapping.* Table II demonstrates the effectiveness of DAG mapping described in Section 2.4. Column 2-4, 5-7 contain the results obtained by DAG mapping with delay minimization objective and area minimization objective, respectively. For each objective, the number of transistors, the number of gates, and delay are reported. Column 4 shows the delay reduction compared to the basic tree-by-tree mapping with the same delay minimization objective. Column 7 shows the area reduction compared to the tree-by-tree mapping with the same area

minimization objective. From the results, we can see that DAG mapping is very effective in reducing the delay, as well as reducing the area.

5.2 Dual-monotonic gate mapping

Table III. Dual-monotonic mapping

Circuits	Dual-mono delay-minimization			Dual-mono area-minimization			#XOR
	Delay	#T/#G	Δ Delay	#T/#G	Delay	Δ Area	
b9	0.15	255/29	0.0 %	243/27	0.18	0.0 %	0
c8	0.17	291/35	0.0 %	274/31	0.21	0.0 %	0
count	0.36	327/37	0.0 %	321/36	0.36	0.0 %	0
i6	0.60	1065/109	0.0 %	763/76	0.62	0.0 %	0
C880	0.71	1063/114	0.0 %	1051/113	0.74	6.7 %	16
C1355	0.33	1628/142	13.2 %	1336/116	0.36	15.2 %	72
C1908	0.47	1727/168	17.5 %	1459/135	0.63	13.8 %	50
C2670	0.51	1885/180	0.0 %	1804/168	0.69	5.7 %	40
C3540	0.72	4368/452	7.8 %	4105/420	0.89	2.5 %	38
C6288	1.63	12323/1250	24.9 %	10942/1187	2.01	2.5 %	419
C7552	0.82	6822/632	4.7 %	6763/625	0.94	8.3 %	213
t481	0.46	1694/201	0.0 %	1611/188	0.62	-0.5 %	0
rot	0.41	1749/199	2.4 %	1681/187	0.52	0.0 %	1
dalu	0.73	2293/240	1.4 %	2239/232	0.86	0.0 %	3
k2	0.50	2688/324	0.0 %	2548/302	0.65	0.0 %	0
des	1.75	10159/1092	0.0 %	9392/930	1.79	1.0 %	51

Table III shows the results of dual-monotonic gate mapping. Columns 2-4 contain the results of the delay minimization objective, and Columns 5-7 show the results of the area minimization objective. Both results were obtained using the dual-monotonic mapping algorithm of Section 3. For each result, both delay and area including the transistor count and gate count are reported. Columns 4 and 7 show the delay reduction compared to the corresponding delay results of the DAG mapping algorithms shown in Columns 2 and 5 of Table II, respectively. Compared with the basic tree-by-tree mapping results shown in Table 11, our dual-monotonic gate mapping results show even larger improvements. We observe that in circuits "t481", our dual-monotonic gate mapping result is a little worse than the corresponding DAG mapping results. This is because of our heuristic approach of collapsing an XOR gate into an XOR node at an early stage, so that an XOR node is always mapped to a dual-monotonic gate during the mapping procedure. A consequence of mapping each XOR node to a merged dual-monotonic gate is that it prevents each polarity of an XOR node from forming an even larger gate with their individual parents. Overall, from the results, we can see that the dual-monotonic gate mapping procedure is quite effective in the circuits with significant XOR substructures.

5.3 Effectiveness of 0-1 output phase assignment

Table IV shows the efficacy of our 0-1 programming output phase assignment optimization algorithms. Here, we use the linear program solver *lp_solve*_2.3 [Berkelaar 1998] to solve the 0-1 integer linear programming formulations. To demonstrate

Table IV. Output phase assignment using a 0-1 ILP: W=8, H=2

Circuits	#po	Orig	Dup-minimize		Cost-minimize		Reduction
		#Tran	#Tran	CPU(s)	#Tran	CPU(s)	%
b9	21	391	389	0.1	311	0.1	20.5%
c8	18	407	364	0.1	330	0.1	18.9%
i6	67	1298	1281	0.1	766	0.1	41.0%
C880	26	1453	1453	0.1	1443	0.1	0.7%
C1355	32	2064	2064	0.1	2064	0.1	0.0%
C1908	25	2268	2268	0.1	2208	0.1	2.6%
C2670	140	2647	2624	0.1	2414	0.1	8.8%
C3540	22	5612	5612	0.1	5552	0.1	1.7%
C6288	32	14257	14257	0.1	14252	0.1	0.0%
C7552	108	9069	9069	0.1	8904	0.1	1.8%
rot	107	2332	2296	0.1	2196	0.1	5.8%
dalu	16	3020	2752	0.4	2745	0.7	9.1%
k2	45	3974	-	-	-	-	-
des	245	13130	13130	0.1	11710	0.2	10.8%
apex7	37	833	791	0.1	736	0.1	11.6%
frg1	3	362	362	0.1	362	0.1	0.o%
x1	35	861	927	0.1	850	0.3	1.3%
x3	99	2381	2360	0.1	2100	0.1	11.8%

the influence of cost difference between two polarities to total implementation cost, the domino gates were restricted to $W = 8$ and $H = 2$.

In Table IV, Column 2 contains the number of primary outputs in the circuits. Column 3 lists the results in the absence of the output phase assignment step. The results obtained by the application of the 0-1 formulations of Sections 4.3 and 4.5 are shown in Columns 4 and 6, respectively and the corresponding CPU times for the linear solver are shown in Columns 5 and 7. Column 8 shows the area reduction due to the output phase assignment optimization considering both duplication minimization and polarity cost difference. Because the output phase assignment has minor influence on the number of levels of the Boolean network, only area costs are listed in Table IV.

The execution time of output phase assignment optimization consists of two parts: cost estimation and linear program solution. Here, we perform the simple tree-by-tree mapping procedure to obtain the cost estimation of each polarity of each tree in the DAG network. The actual running time for this step is much lower than Column 3 of Table 1 since a large part of the CPU time there is spent on parsing, data structure building and unating. The CPU time required by the linear solver is listed in Table IV. We can see that all of the benchmarks can solved in under a second with no or minor simplification of the 0-1 ILP. An exception is the 0-1 programming problem for Circuit $k2$, which does not yield a result in reasonable time. The increase in the size of the linear program causes a minor increase in CPU time in column 7 as compared to column 5.

It was observed that in some circuits such as *dalu*, the cost reduction by output phase assignment arises mainly from the duplication cost reduction, while in some circuits such as $i6$ and $b9$, the implementation cost difference between two polarities becomes the most significant consideration for output phase assignment. Interestingly, we see that in some cases, the two objectives of output phase assignment can

be contradictory to each other. In Circuit $x1$, the optimization for duplication cost minimization causes a cost increase due to the increased cost of implementing the reversed polarity and increases the total implementation cost.

Table V. W=8, H=2 vs. W=6, H=3

Circuits	W=6, H=3				W=8, H=2
	No-ass	Dup-m	Cost-m	Reduction	Reduction
b9	291	284	249	14.4%	20.5 %
c8	322	314	280	13.0%	18.9 %
i6	958	951	761	20.6%	41.0 %
C880	1153	1153	1153	0.0%	0.7 %
C1355	1834	1834	1834	0.0%	0.0 %
C1908	2038	2038	2023	0.7%	2.6 %
C2670	2177	2149	2094	3.8%	8.8 %
C3540	4587	4587	4562	0.6%	1.7 %
C6288	13707	13707	13707	0.0%	0.0 %
C7552	8069	8069	8039	0.4%	1.8 %
rot	1867	1846	1796	3.8%	5.8 %
dalu	2500	2365	2365	5.4%	9.1 %
k2	3174	-	-	-	-
des	10685	10685	10115	5.3%	10.8 %
apex7	668	646	636	4.8%	11.6 %
frg1	287	287	287	0%	0.0 %
x1	691	722	685	0.9%	1.3 %
x3	1906	1905	1795	5.8%	11.8 %

From Table IV, we can see output phase assignment optimization considering the cost difference between two polarities plays an important role in the cost reduction in case of $W = 8$ and $H = 2$. However, the less the difference between W and H, the less is the cost difference between the two polarities and the lower is the cost reduction benefit from the output phase assignment for the polarity difference factor. In Table V, the output phase assignment results under the constraint of $W = 6$ and $H = 3$ are shown in Column 2-5. Compared with the cost reduction under the constraint of $W = 8$ and $H = 2$ in Column 6, we can see that the ratio of cost reduction reduced, as expected. Here, Column 6 of Table V is same as the Column 8 of Table IV.

5.4 Comparison with technology mapping of static logic

Table VI shows a comparison our DAG domino mapper with technology mapping to a static CMOS library using SIS. For fairness, the experiments here compare the use of $W = H = 4$ for the domino implementation with the adapted static library 44-3.genlib, which includes all static gates with up to four series transistors and four parallel chains. The objective for both mappers was delay minimization.

Note that to obtain the homogeneous comparison between SIS delay evaluation and domino logic delay evaluation, the library $44-3.genlib$ used here was adapted to apply the same delay and area model as our domino mapper. In the adapted library $44-3.genlib$, the cell areas of library $44-3.genlib$ were replaced with the transistor count and the same Elmore delay model as Formula 1 in Section 2.3.4 are used for the static rising and falling delay of the static cells. These Elmore delay

Table VI. A comparison of our domino mapping algorithm with SIS

Circuits	Our domino mapper			SIS (44-3.genlib)			Dup-ratio
	Delay	#Tran	#PMOS	Delay	#Tran	time(s)	%
b9	0.15	255	58	0.35	392	11.5	10%
c8	0.17	291	70	0.39	374	12.6	24%
count	0.36	327	74	0.86	420	12.5	22%
i6	0.60	1065	218	0.98	1118	50.3	13%
C880	0.71	1063	248	1.57	1290	40.9	47%
C1355	0.33	1628	304	0.75	1662	36.8	77%
C1908	0.47	1727	402	1.15	1666	39.9	74%
C2670	0.51	1885	382	0.97	2298	70.7	58%
C3540	0.72	4368	932	1.66	3644	133.7	92%
C6288	1.63	12323	2100	4.06	11200	228.0	97%
C7552	0.82	6822	1522	3.37	7362	178.6	79%
t481	0.46	1694	402	0.80	2204	82.9	6%
rot	0.41	1749	398	0.91	2044	57.0	33%
dalu	0.73	2293	478	1.70	2854	99.4	43%
k2	0.50	2688	648	0.90	3420	97.0	2%
des	1.75	10159	2184	7.38	10642	178.6	49%

formulas are then fitted into the SIS delay model to obtain the *rise-block-delay*, *rise-fanout-delay*, *fall-block-delay*, *fall-fanout-delay* of each cell. In the static delay model, the size of each PMOS transistor are assumed to be twice the size of NMOS transistor.

The entries in the 2-4 column show the results of our DAG domino mapper in terms of delay, the number of transistors and the number of PMOS transistors. Columns 5-7 list the delay, the number of transistors and CPU time of the SIS static mapper. In the last column, we report the ratio of nodes that must be duplicated in comparison with the number of nodes in the unated circuits.

Empirically, it has been observed that domino logic runs $1.5-2\times$ faster than static CMOS logic [Harris and Horowitz 1997] at about the same area as static CMOS logic, and comparisons of static and domino logic implementations are available in [Harris and Horowitz 1997; Yee and Sechen 1997; Prasad et al. 1997; Thorp et al. 1998]. From our results listed above, we can see that the mapped domino circuits, especially through the DAG domino mapper, has much better delay performance than its static counterpart. The area cost of the domino circuits is smaller than or close to that of the corresponding static circuit for the small and intermediate sized benchmarks. In large circuits, with the increase of the depth of the circuit, the number of nodes that need to be duplicated becomes larger. Therefore, the area cost of domino logic is larger than that of the corresponding static implementation. In the situation where the amount of duplication is larger, the static implementation has the advantage in terms of area cost.

In Table VI, the transistor count and the PMOS count are listed for both static and domino logic (half of the transistors of static circuits are PMOS transistors). It must be noted that these are just coarse measurements of the area cost. There are other factors that need to be considered for the more accurate area cost comparisons. Firstly, compared with static circuits, domino logic uses a very small fraction of PMOS transistors, which are typically sized to be larger than NMOS transistors.

Secondly, the flexible clock scheme of domino logic usually splits the large combinational circuit into smaller circuits associated with different phases, which may reduce the duplication cost [Zhao and Sapatnekar 2000]. However, there are also several disadvantages in terms of area for domino implementation. Usually a keeper and the precharge devices have to be added to each domino gate. In addition, there are larger overheads for clock routing, and the low noise margins of domino logic may require additional layout area due to extra shielding tracks and wider spacing.

6. CONCLUSIONS

In this paper, we have mainly explored technology mapping techniques for domino logic. Based on principles of dynamic programming, an efficient parameterized library mapping algorithm is presented. Several other technology mapping methods considering various features of domino logic, such as DAG mapping, dual-monotonic mapping and output phase assignment, are proposed and incorporated in the mapper.

Our mapper can be applied to objective of delay minimization, area minimization, as well as area minimization mapping under timing constraints. The results of our work have been shown to provide a smaller area/delay than the previous work and have low CPU time. The comparisons of the enhancement methods with basic method indicate the effectiveness of several methods proposed in this paper. Compared to static CMOS mapping, domino logic synthesis results have better delay performance. The area overhead for domino circuits is close to or better than the cost of a static implementation, depending on the duplication ratio.

References

Benini, L. and Micheli, G. D. 1997. A survey of boolean matching techniques for library binding. *ACM Transactions on Design Automation of Electronic Systems 2*, 3 (July), 193–226.

Berkelaar, M. R. C. M. 1998. LP SOLVE 2.3 Users' Manual (1998).

Berkelaar, M. R. C. M. and Jess, J. A. G. 1988. Technology mapping for standard-cell generators. In *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design* (1988). 470–473.

Brayton, R. K., Chen, C. L., McMullen, C. T., Otten, R. H. J. M., and Yamour, Y. J. 1984. Automated implementation of switching functions as dynamic CMOS circuits. In *Proceedings of the IEEE Custom Integrated Circuits Conference* (1984). 346–350.

Burns, J. L. and Feldman, J. A. 1998. C5M- a control-logic layout synthesis system for high-performance microprocessors. *IEEE Transactions on Computer-Aided Design 17*, 1 (Jan.), 14–23.

Chaudhary, K. and Pedram, M. 1995. Computing the area versus delay trade-off curves in technology mapping. *IEEE Transactions on Computer-Aided Design 14*, 12 (Dec.), 1480–1489.

Cormen, T. H., Leiserson, C. E., and Rivest, R. L. 1990. *Introduction to Algorithms*. McGraw-Hill, New York.

D. V. Campenhout, Mudge, T., and K.Sakallah. 1996. Modeling domino logic for static timing analysis. Technical Report CSE-TR-295-96, The University of Michigan.

Detjens, E., Gannot, G., Rudell, R., Sangiovanni-Vincentelli, A., and Wang, A. 1987. Technology mapping in MIS. In *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design* (1987). 116–119.

Fishburn, J. P. and Dunlop, A. E. 1985. TILOS: A posynomial programming approach to transistor sizing. In *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design* (1985). 326–328.

Harris, D. and Horowitz, M. A. 1997. Skew-tolerant domino circuits. *IEEE Journal of Solid-State Circuits 32*, 11 (Nov.), 1702–1711.

Hoffman, C. M. and O'Donnell, M. J. 1982. Pattern matching in trees. *Journal of the Association for Computing Machinery 29*, 1 (Jan.), 68–95.

Jongeneel, D., Otten, R., Watanabe, Y., and Brayton, R. K. 2000. Area and search space control for technology mapping. In *Proceedings of the ACM/IEEE Design Automation Conference* (2000). 86–91.

Keutzer, K. 1987. DAGON: technology mapping and local optimization. In *Proceedings of the ACM/IEEE Design Automation Conference* (1987). 341–347.

Kim, K.-W., C.L.Liu, and Kang, S.-M. 1999. Implication graph based domino logic synthesis. In *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design* (1999). 111–114.

Kukimoto, Y., Brayton, R. K., and Sawkar, P. 1998. Delay-optimal technology mapping by DAG covering. In *Proceedings of the ACM/IEEE Design Automation Conference* (1998). 348–351.

Lehman, E., Watanabe, Y., Grodstein, J., and Harkness, H. 1997. Logic decomposition during technology mapping. *Journal of the Association for Computing Machinery 16*, 8 (Aug.), 813–834.

Matsunaga, Y. 1998. On accelerating pattern matching for technology mapping. In *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design* (1998). 118–123.

Micheli, G. D. 1994. *Synthesis and optimization of digital circuits*. McGraw-Hill, Inc., New York, NY.

Ortiz, R. R. and Lefebvre, M. C. 1993. Technology mapping for NORA dynamic logic circuits. In *Proceedings of the European Design Automation Conference* (1993). 310–314.

Patra, P. and Narayanan, U. 1999. Automated phase assignment for the synthesis of low power domino circuits. In *Proceedings of the ACM/IEEE Design Automation Conference* (1999). 379–384.

Prasad, M. R., Kirkpatrick, D., and Brayton, R. K. 1997. Domino logic synthesis and technology mapping. In *Workshop Notes, International Workshop on Logic Synthesis* (1997).

Puri, R. 1998. Design issues in mixed static-domino circuit implementations. In *Proceedings of the IEEE International Conference on Computer Design* (1998). 270–275.

Puri, R., Bjorksten, A., and Rosser, T. E. 1996. Logic optimization by output phase assignment in dynamic logic synthesis. In *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design* (1996). 2–8.

Sapatnekar, S. S. and Kang, S. M. 1993. *Design automation for timing-driven layout synthesis*. Boston, MA: Kluwer Academic Publishers.

Thorp, T., Yee, G., and Sechen, C. 1998. Domino logic synthesis using complex static gates. In *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design* (1998). 242–247.

Touati, H. J., Moon, C. W., Brayton, R. K., and Wang, A. 1990. Performance-oriented technology mapping. In *MIT Conference on Advanced Research in VLSI* (1990). 79–97.

Venkat, K., Chen, L., Lin, I., Mistry, P., and Madhani, P. 1996. Timing verification of dynamic circuits. *IEEE Journal of Solid-State Circuits 31*, 3 (Mar.), 452–455.

Wang, J., Wang, Z. D., Jullien, G. A., and Miller, W. C. 1994. Area-time analysis of carry lookahead adders using enhanced multiple output domino logic. In *Proceedings of the IEEE International Symposium on Circuits and Systems* (1994). 59–62.

Wang, Z., Jullien, G. A., Miller, W. C., Wang, J., and Bizzan, S. S. 1997. Fast adders using enhanced multiple-output domino logic. *IEEE Journal of Solid-State Circuits 32*, 2 (Feb.), 206–213.

Williams, T. 1996. Dynamic logic: Clocked and asynchronous. Tutorial notes at the International Solid State Circuits Conference.

Yee, G. and Sechen, C. 1997. Dynamic logic synthesis. In *Proceedings of the IEEE Custom Integrated Circuits Conference* (1997). 345–348.

Zhao, M. and Sapatnekar, S. S. 1998. Technology mapping for domino logic. In *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design* (1998). 248–251.

Zhao, M. and Sapatnekar, S. S. 2000. Timing-driven partitioning and timing optimization of mixed static-domino implementations. *Journal of the Association for Computing Machinery 19*, 11 (Nov.), 1322–1336.