Timing-driven Partitioning for Two-Phase Domino and Mixed Static/Domino Implementations *

Min Zhao and Sachin S. Sapatnekar
ECE Department, University of Minnesota, Minneapolis MN 55455, USA.

Abstract

Domino logic is a high-performance circuit configuration that is usually embedded in static logic environment and tightly coupled with the clocking scheme. In this paper, the timing-driven partitioning algorithms that partition a logic network between (1) static and domino implementations, and (2) the phases of a two-phase clock, are provided. In addition, an efficient static mapping algorithm is described.

1 Introduction

Domino logic is an effective circuit configuration for implementing high-speed logic designs [1, 2]. Domino synthesis has been an active area of recent research. In [3], a domino logic synthesis flow including logic optimization and technology mapping is described, and in [4, 5], domino gate synthesis methods are described. The work in [6] addresses 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 work presents novel domino clock schemes, such as methods for overcoming the noninverting property of domino logic [7], and making the design more skew tolerant [2].

Both static and domino implementations offer various advantages. For example, domino circuits typically provide higher speeds than static logic, but have a higher clock routing overhead and are less noise-tolerant. Depending on the amount of logic duplication required to make the network unate, domino circuits may need a larger or smaller number of transistors than static circuits. Therefore, for optimality, a determination must be made as to which parts of the circuit should be implemented in static logic, and which parts using domino logic. On the other hand, since synchronous domino logic is conventionally divided into two phases, the partitioning of domino gates between clock phases is essential for correct and efficient performance of domino gates.

In this paper, we describe an automated design strategy to solve problems related to static-domino partitioning and two-phase domino partitioning, with the goal of minimizing an objective (area or power) under a set of timing constraints. Due to space limitations, we will only discuss using area as a cost function. Our partitioning procedures are tightly knit with an efficient technology mapping algorithm, so that the results provide optimality in the cost function after technology mapping.

A related paper [8] presents theory on a partitioning framework without any experimental results. Our procedure is based on the following observations that differentiate it from [8]:
(a) Logic duplication can cause a large area penalty for large combinational circuits [3]. As described in Section 2, a proper choice of the partitioning cut can reduce the duplication cost. Our partitioning procedure automatically creates the largest possible unate region within the domino partition.
(b) The critical path and its fanin transition cone may form a large part of the input, or possibly even the entire network (for a circuit with one primary output). As long as the timing constraints are satisfied, greedily maximizing the use of domino gates in this cone as in [8] may be unnecessary, especially if it has a high cost.

As a part of this work, we report an extension to a published domino technology mapper [5] to perform static mapping, and find that this performs average 3.8% better than SIS, while being over two orders of magnitude faster.

*This work was supported in part by the Semiconductor Research Corporation under grant 99-TJ-692 and by a grant from Intel Corporation.

2 Partitioning considerations

We define the above two problems concisely as follows:
(1) **Static-domino partitioning**: Given a combinational circuit and delay specifications on the outputs of the network, implement the nodes in the network using either domino logic gates or static gates such that the cost is minimized, assuming them all to be within the *same* clock phase. For correct circuit operation, this should satisfy a precedence constraint that states that no static logic gate is permitted to fan out to a domino gate. The timing constraint for this situation is that the partitioned circuit must have its outputs ready at the end of the clock phase.
(2) **Two-way domino partitioning**: Given a combinational circuit to be implemented entirely in domino and a two-phase nonoverlapping clock scheme, partition the boolean network into two clock phases such that the cost of the implementation is minimized. Here, the latch between the two phases of domino forms a hard edge that prevents cycle borrowing [2, 9], implying a timing constraint that the delay is less than a half-cycle.

The timing driven partitioning method must consider two factors: the timing constraints and the area of the implementation. In our description, we will primarily refer to the area as the cost, but the power may be considered equivalently. An important consideration in the cost relates to the requirement that only unate functions can be implemented in domino logic, due to the noninverting nature of the domino logic family [6]. However, we observe that the duplication cost required to maintain this property can be reduced by partitioning, and our formulation directly incorporates the cost of duplicating nonunate logic within a domino region in the cost function.

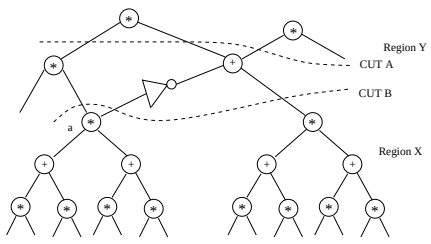

Figure 1: An example for static-domino partitioning

To see this, consider the example shown in Figure 1. Let Region X correspond to a domino implementation and Region Y

represent a static CMOS implementation. If cut A is used, then both a and $\overline{a}$ must be implemented as unate functions, resulting in a duplication of the fanin cone of a, doubling the cost of implementing this cone. However, if cut B is used instead, then there is no need to duplicate the fanin cone of a as the inverter may be implemented in static logic. A similar statement can also be made for the two-way domino partitioning problem.

3 Timing driven static-domino partitioning

The two subproblems listed in the previous section can be solved using similar algorithms, using different cost formulations. The partitioner applies technology mappers, a PERT based timing analysis technique and a maximum flow algorithm to realize a timing-driven two-way partitioning. The input to the algorithm is an arbitrary two-input AND-OR DAG network. The outline of the algorithm is as follows:

(1) Perform static technology mapping and domino technology mapping separately on the entire logic network to determine a cost estimate for every vertex.

(2) Find the candidate cut nodes in the network. A candidate cut node is defined as a node that satisfies the criterion that a cut passing through it will not violate the timing specification.

(3) Build the flow network from the candidate cut nodes. The edge capacities are determined from the cost difference between static and domino implementations from Step 1. A maximum flow algorithm is applied to the network to obtain a mincut [10].

An incremental transistor level Elmore delay model is developed to fit the cost function calculation of technology mapping algorithms. Details are omitted due to space limitations.

3.1 Cost estimation

The first step of partitioning is to perform static and domino mapping on the logic network to obtain the cost estimation. A cost comparison is then carried out for each vertex under a domino and a static circuit implementation. Any static and domino mappers may be used; the ones used here are described in Section 6.

The task of domino mapping is complicated by the fact that technology mapping of domino logic requires the input network to be unated first. To address the issue, we introduce a *twin network*, N_{twin},to represent the original network, on which domino mapping for unated network can be performed while no explicit duplication is required. Each vertex in N_{twin} corresponds to a node in the original network, and stores information on the implementation of the true and complemented forms of the logic function realized at that node; these will be referred to as the positive and negative polarities, respectively. An edge between two nodes can have two polarities: if an inverter exists between these nodes in the original network, then the edge polarity in N_{twin} is negative; otherwise, it is positive. Inverters in the original network may be merged into the edge polarities. The lists of positive and negative polarity fanouts of each vertex in N_{twin} are maintained, and for domino mapping any fanout duplication is flagged.

Static mapping can also be performed on N_{twin}. The cost estimation at each vertex from technology mapping includes both delay and area information, and the quantities $delay_d$, $area_d$, $delay_s$ and $area_s$ are calculated at each node under both positive and negative polarities, corresponding to the delay and fanin cone area estimate under domino and static mappings, respectively.

3.2 Determining the candidate cut nodes

We introduce the idea of a *candidate cut node* using a method that eliminates vertices that would violate timing constraints after a cut. Given a vertex v in N_{twin}, if the cutset passes through v, its input transition cone will be implemented by domino gates and the output transition cone of v will be implemented by static gates. Let $D_d(i,v)$ be the largest delay from the inputs to node v using a domino implementation, and $D_s(v,o)$ be the largest delay from the node v to outputs through paths using static logic. Then the maximal delay from the input to output that passes through the cut at node v will be $D_d(i,v)+D_s(v,o)$. If this value is smaller than the specified delay, T_{spec}, then the cut through node v is eligible; if not, it is certain to violate the timing constraint. The value of $D_s(v,o)$ for all the nodes is obtained from one PERT-like traversal of the network from the outputs towards its inputs after technology mapping, and $D_d(i,v)$ is obtained directly as a result of technology mapping.

3.3 Finding the minimum cut

For domino partition problems, in addition to the partitioning cost, implementation cost itself varies (due to the duplication penalty) with partitions. To solve the problem, a maximum flow network is built using these nodes using an identical DAG structure as in the original circuit, with capacities assigned to the edges. The maxflow mincut algorithm is then applied to this network to find the minimum cut.

Maximal flow capacity assignment

During technology mapping (performed using DAG mapping rather than tree mapping), the area contribution of any multifanout node is divided by the fanout count of the node as in [11], so that its area value, $area(N)$, estimates the area contribution of its fanin transition cone. Therefore, for any cut on the graph, the area cost in the region from the primary inputs to the cutset can be estimated as $\sum_{i\in CutSet} area_d(N_i)$, the minimum area over either polarity in domino.

Consider a cut that divides the network into two parts, referred to as Region X (closer to the inputs) and Region Y (closer to the outputs) as in Figure 1. Assume the area cost of the entire network implemented in static logic is $A_s(sum)$, and the costs for Region X to be implemented in static and domino logic are, respectively, $A_s(X)$ and $A_d(X)$. Then the total cost after partitioning, with Region X implemented in domino logic and Region Y implemented with static gates, is $A_s(sum)-A_s(X)+A_d(X)$. Removing the constant, $A_s(sum)$, the objective of minimizing cost is equivalent to minimizing $-A_s(X)+A_d(X)=\sum_{i\in cutset}(area_d(i)-area_s(i))$.

Based on this, we reduce the problem to one of finding a minimum cost vertex cut on the network. The capacity associated with each node i is set to $area_d(i)-area_s(i)$. The vertex cut must maintain the predecessor constraints that dictate that no static node can feed a domino node.

Building the maximum flow graph

The procedure of building the maximum flow network can be illustrated on the example circuit of Figure 1. The circuit is translated into a topology of the type shown in Figure 2(a). The shaded part of the network shows the region containing the candidate cut nodes. The two weights on each node represent $area_d(i)$ and $area_s(i)$, the result of domino and static mapping, respectively.

Finding the minimum cost cut of the above network is equivalent to finding the minimum cost cut on the maximum flow graph shown in Figure 2(b). The nodes in the shaded region that are closest to the primary inputs are connected to the source, and the nodes of the region that are closest to the primary outputs are connected to the sink. Each vertex in Figure 2 is split into two vertices connected by an edge of capacity $area_d(i)-area_s(i)$.

However, before the standard maxflow mincut algorithm can be applied on the network, two conditions in the network must be considered: (1) The vertex cut must maintain the predecessor constraints that dictate that no static node can feed a domino

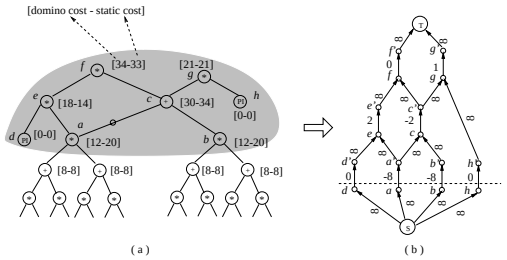

Figure 2: (a) N_{twin} to be partitioned (b) Its vertex cut network

node. The solution to this problem is provided in the work in [12, 13] if all edge capacities are positive. (2) Our network can have negative edge capacities, which complicates the previous issue. Moreover, standard maximum flow algorithms cannot handle edges with negative capacities, and the network must be modified suitably. To overcome these problems, we heuristically transform the vertex-cut maxflow network into an edge-cut maxflow network, and then and translate it into a standard maxflow network with nonnegative edge capacities. The procedure is as follows:
(1) The edge cut maximum flow network is built. If (u,v) is an edge originating at the candidate cut node u in N_{twin}, the capacity of the edge is heuristically assigned to $C_{init} = (area_d(u) - area_s(u))/\text{fanout}(u)$, where fanout$(u)$ is number of fanouts of u.
(2) A positive initial flow is injected into the source node, and the initial flow is distributed into the whole network. The flow at each node is calculated by a PERT-like traversal on the DAG, with the flow from node u to a fanout node v being calculated as $Flow(u,v) = \sum_{i \in input(u)}(Flow(i))/\text{fanout}(u)$. Since this is a feasible flow, updating the capacity C of each edge as $C_{new}(uv) = C_{init}(uv) + Flow(u,v)$ leaves the identity of the minimum cost cut unchanged. This procedure is repeated by increasing the value of the initial flow at the source node until the value of C_{new} for each edge is nonnegative.
(3) For each edge (u,v) in the graph, a new edge (v,u) with a capacity of ∞ is introduced into the graph to force the predecessor constraint. The maxflow mincut algorithm is then applied to the network to obtain the minimum cost cut.

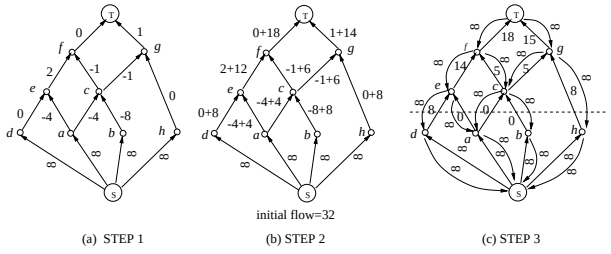

Figure 3: Constructing the edge-cut maximum flow network

4 Two-way domino partitioning

The problem of two-way domino partitioning of a circuit under a two-phase clock is to determine which gates in the circuit should be clocked by the first clock phase, and which by the second clock phase. The algorithms are largely similar, and the differences between them are summarized as follows:
(1) **Determining the candidate cut nodes**: For any vertex N of the input network, let $D_d(i,N)$ be the largest delay from the inputs to node N and let $D_d(N,o)$ be the largest delay from node N to the outputs, calculated using a reverse PERT traversal as before. The physical meaning of the cut in this situation is that if the cutset passes through some node N, then all gates in the fanin transition cone of N will be clocked by clock phase Φ_1, and all gates in the fanout transition cone of N will be clocked by Φ_2. Therefore, a vertex is a candidate cut node if both $D_d(i,N)$ and $D_d(N,o)$ are smaller than the clock pulse width.
(2) **Maximal flow capacity**: If the two clock phase regions are separated by a latch that can internally invert a signal, then the cost function is the sum of the latch costs and the reduction in the cost of logic duplication. Assume that N is a node at which both the true and complemented form of the logic function are required. If N does not belong to the cutset, the logic must be duplicated, and the area cost at this node is the sum of $area_d(N)$ for both polarities. If the node lies on the cut, only one polarity of the logic needs to be generated and the other is generated within the latch. Therefore, the area contribution as the smaller of $area_d$ over the two polarities, and the area cost difference between passing a cut through N or not is the $area_d$ of the other polarity, plus the latch area. If only one polarity is needed for a node N, the area cost difference for N is merely the area of the latch.

5 General static-domino partitioning

Using the solutions developed in previous sections, we solve the problem under a general two-phase nonoverlappingscheme [8, 9]. We present two design flows, the first of which is **Flow 1** below:
(1) We first perform static-domino partitioning on the entire network to divide as Region 1 (domino) and Region 2 (static). The timing constraint here is a full clock cycle.
(2) We now specify the required time at the output of Region 1. Next, we perform two-way domino partitioning to obtain the region to be (tentatively) clocked by phase Φ_1 (Region 3) and the region to be clocked by phase Φ_2 (Region 4).
(3) Region 3 is now assigned to Phase 1, but we may now choose to implement a part of it in static logic. Therefore, we perform static-domino partitioning on this region to obtain a phase Φ_1 domino region (Region 5) and a phase Φ_1 static region (Region 6). The result of this procedure is that Region 2 and Region 6 are implemented in static CMOS, Region 5 is implemented as phase Φ_1 domino, and Region 4 as phase Φ_2 domino.

Another possible partitioning design flow (**Flow2**) first performs two-way domino partitioning on the input circuit, followed by static-domino partitioning on the two partitions.

Table 1: Comparison of the library mapping vs. SIS results

Circuits	lib-free 4-4	CPU (s)	SIS 44-6	CPU (s)	lib-free 4-4	SIS 44-6
C1355	1352/17	0.27	1378/20	108	962	978
dalu	2106/22	0.67	2284/24	278	1330	1497
C880	1024/19	0.21	1046/33	114	688	711
count	352/31	0.06	324/17	36	240	212
C1908	1324/30	0.27	1344/33	112.7	941	960
C2670	1770/20	0.56	1936/17	202.1	1188	1356
C3540	2820/34	0.94	3112/35	368	1864	2076
C6288	8388/107	3.24	8728/104	662	6016	6400
k2	2852/19	0.69	2910/23	254.4	2024	2101
des	8226/18	3.16	8628/19	1030	5337	5728
C7552	5546/25	2.14	5744/26	537	3768	3942
rot	1668/15	0.45	1714/20	167.2	1162	1214

6 Static technology mapping

The procedure in Section 3 requires technology mapping to fully static and fully domino circuits. For domino circuits, we use the algorithm in [5], and for static circuits, we describe an extension of the same algorithm here. We will see that this mapper can be orders of magnitude faster than the SIS mapper and gives a quality improvement of 0 to 12.3% over SIS.

The basis for this work lies in [5], where a modified dynamic programming algorithm using parameterized library mapping was

Table 2: Static-domino and two-way domino partitioning

Circuits	Domino 4/4	Static N_{tran}/delay	Static-domino partitioning							Two-way domino partitioning		
			N_{tran}(no spec)	G_d/G_s	$N_{tran}(\times 1.25)$	G_d/G_s	$N_{tran}(\times 1.05)$	G_d/G_s	CPU(s)	N_{tran}(no spec)	$N_{tran}(\times 1.05)$	CPU(s)
c1355	1824	1302/×2.25	1302	0/260	1800	170/104	1800	170/104	1.4	1656	1600	1.8
dalu	2360	2192/×2.16	2098	97/198	2096	147/132	2096	189/75	7.9	1971	2080	10.0
c880	1163	982/×2.08	958	21/124	1015	56/88	1027	62/85	1.4	933	1037	4.6
count	357	336/×2.77	344	5/54	350	23/30	353	32/18	0.3	267	347	0.4
c1908	1978	1308/×1.78	1306	5/263	1723	174/86	1928	238/34	1.4	1867	1838	2.7
c2670	1992	1754/×1.75	1775	79/173	1775	79/173	1774	81/170	3.5	1703	1705	6.0
c3540	4527	2850/×1.43	2748	88/349	3312	260/218	3987	461/26	10.9	3499	3499	11.7
c6288	13702	8350/×1.78	8340	16/1771	12079	1301/493	13456	1733/73	33.5	13173	13170	96.4
k2	2884	2896/×1.54	2884	368/68	2884	368/68	2884	368/68	8.6	2856	2920	9.6
des	9945	8134/×4.25	7527	160/915	7536	165/911	7536	165/911	60.2	8265	10835	111.5
c7552	7919	5464/×2.35	5370	78/296	5987	375/578	6198	456/504	30.9	6434	6607	44.6
rot	1777	1536/×1.99	1462	55/171	1514	87/137	1611	126/103	3.0	1422	1638	4.9

proposed for domino logic. The parameterized library considers all possible cells that have at most W transistors in parallel and H transistors in series. The differences between this method and the domino mapping method are as follows. Firstly, for each vertex of N_{twin}, subsolutions corresponding to both positive and negative logic polarities are maintained. This permits the free (virtual) movement of inverters throughout the network to allow a better exploration of the design space. We map the largest possible logic cone in the network at a time. Next, we find its optimal implementation and assign a polarity to each node; this polarity for multifanout nodes is then fixed and used to map the remaining logic cones. Secondly, the cost function can either be taken to be the number of transistors or more general mapping functions specified by a lookup table characterized by the subsolution parameters of its child nodes. A comparison between this algorithm with results of SIS is shown in Table 1 using $H = W = 4$. The complete library under $H = W = 4$, 44-6.genlib,is used for SIS technology mapping, with the objectives of both algorithms being area minimization. Two sets of comparison between our mapper and SIS are shown. The first set uses the number of transistors as the area cost model, with the library 44-6.genlib being altered so that the cell area equals the transistor count. The area cost and CPU time for both our parameterized library mapper, and the SIS technology mapper are shown in columns 2 to 5.

The second set uses the cost model of library 44-6.genlib, and the cell area assignment of 44-6.genlib is fitted into the parameterized library cost model. Columns 6 and 7 show the results of our library mapper and SIS, respectively, using this cost model.

7 Experimental results

The partitioning and technology mapping flow has been implemented in C++. The maximum number of series/parallel transistors in any gate for the parameterized library is set to be 4/4 and 3/3 for domino and static gates, respectively, and the objective is set to area minimization. *script.rugged* in SIS is used for initial logic minimization on all circuits.

The results of static-domino partitioning are shown in Table 2. The second and third columns show the results of pure one-phase domino mapping and pure static mapping, respectively. The delay of the static implementation is also shown. All delays here in this table are normalized; "$\times 1.0$" corresponds to the delay of a purely domino implementation. Next, for various delay specifications, we list the number of transistors, N_{tran}, and the number of domino/static gates (G_d/G_s) using our method. From the table, we can see that when the timing constraints are larger, the cost is smaller than the minimum of columns 2 and 3, as expected. We observe that domino implementations of the benchmarks usually have a speed advantage over static circuits, but tend to have larger areas. Therefore, the tighter the timing constraints, the more domino gates are required, and the larger the area cost.

The last three columns show the results of two way domino partitioning under two specifications. In most cases the resulting area is about the same or larger for the tighter specification, as expected. In a few cases, a slightly smaller area is obtained (but within a reasonable margin of error); this is due to approximations in the cost estimation in DAG mapping, which is the main factor that prevents the partition algorithm from attaining optimality.

The differential between N_{tran}(no spec) here and column 2 shows the reduction in logic duplication obtained by exploiting the inverters at the partition boundary. Table 3 shows the results of **Flow 1** and **Flow 2** from Section 5 for a general clock scheme. We observe that **Flow 2** introduces more latches than **Flow 1**.

Table 3: Results of partitioning for a general clock scheme

Circuits	Flow 1 $\times 1.25$	Flow 1 $\times 1.05$	Flow 2 $\times 1.25$	Flow 2 $\times 1.05$
c1355	1408/8	1456/8	1452/48	1486/48
dalu	1998/56	2050/78	1923/63	2049/117
c880	944/13	953/14	926/43	943/43
count	346/9	345/14	337/23	338/23
c1908	1449/46	1560/46	1519/40	1590/60
c2670	1540/60	1538/52	1548/95	1548/95
c3540	3063/60	3235/68	2943/53	3277/53
c6288	11604/104	12511/115	12105/110	12410/111
k2	2691/157	2795/152	2862/147	2889/156
des	7510/118	7513/119	8452/437	8766/437
c7552	5754/164	5772/164	5701/192	5892/194
rot	1463/36	1515/51	1485/118	1538/119

References

[1] P. E. Gronowski, W. J. Bowhill, R. P. Preston, M. K. Gowan, and R. L. Allmon, "High-performance microprocessor design," *IEEE Journal of Solid-State Circuits*, vol. 33, pp. 676–686, May 1998.

[2] D. Harris and M. A. Horowitz, "Skew-tolerant domino circuits," *IEEE Journal of Solid-State Circuits*, vol. 32, pp. 1702–1711, Nov. 1997.

[3] M. R. Prasad, D. Kirkpatrick, and R. K. Brayton, "Domino logic synthesis and technology mapping," *Proc. IWLS*, 1997.

[4] G. Y. T. Thorp and C. Sechen, "Domino logic synthesis using complex static gates," *Proc. ICCAD*, pp. 242–247, 1998.

[5] M. Zhao and S. Sapatnekar, "Technology mapping for domino logic," *Proc. ICCAD*, pp. 248–251, 1998.

[6] R. Puri, A. Bjorksten, and T. E. Rosser, "Logic optimization by output phase assignment in dynamic logic synthesis," *Proc. ICCAD*, pp. 2–8, 1996.

[7] G. Yee and C. Sechen, "Dynamic logic synthesis," *Proc. CICC*, pp. 345–348, 1997.

[8] R. Puri, "Design issues in mixed static-domino circuit implementations," *Proc. ICCAD*, pp. 270–275, 1998.

[9] T. Williams, "Dynamic logic: Clocked and asynchronous." *Tutorial notes at ISSCC*, 1996.

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

[11] K. Chaudhary and M. Pedram, "Computing the area versus delay trade-off curves in technology mapping," *IEEE Trans. on CAD*, vol. 14, pp. 1480–1489, Dec. 1995.

[12] H. Liu and D. F. Wong, "Network flow based circuit partitioning for time-multiplexed fpga's," in *Proc. ICCAD*, pp. 497–504, 1998.

[13] C. F. S. Iman, M. Pedram and J. Cong, "Finding uni-directional cuts based on physical partitioning and logic restructing," *Proc. 4th Int. Workshop on Physical Design*, 1993.