Timing-driven Partitioning and Timing Optimization of Mixed Static-Domino Implementations

Min Zhao and Sachin S. Sapatnekar

Abstract— **Domino logic is a circuit family that is well-suited to implementing high-speed circuits. Synthesis of domino circuits is more complex than static logic synthesis due to the non-inverting nature of the logic and the complex timing relationships associated with the clock scheme. In this paper, we address several problems along a domino synthesis flow. We mainly consider the problem of partitioning a circuit into static and domino regions under timing constraints. The algorithm is extended to develop a method for partitioning domino logic into two phases, with inverters permitted between the two phases, and then to a flow for general two-phase static-domino partitioning. We also address a timing verification and sizing optimization tool for circuits containing mixed domino and static logic.**

Keywords— **Domino logic, logic duplication, partitioning, sizing, static logic, technology mapping, timing, VLSI.**

I. Introduction

Domino logic is an effective circuit configuration for implementing high-speed logic designs. Despite some drawbacks, such as its susceptibility to charge-sharing and noise and its non-inverting property, it has made important contributions in the design of low cycle time microprocessor design and other high performance designs [1], [2]. Domino circuits offer the advantages of faster speed and glitch-free operation.

A representative domino gate configuration is shown in Figure 1. When the clock input is low, the gate precharges 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 two-step mode of operation with a precharge and an evaluate phase causes the timing relationships in domino logic to be more complex than those for static logic.

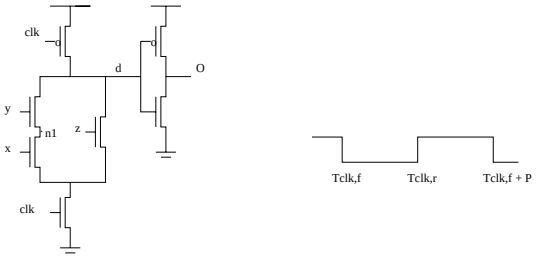


Fig. 1. A typical domino circuit

In the past, most domino circuits have been manually designed and automated synthesis tools for domino logic have been scarce. Recent research has led to several papers related to domino synthesis and optimization. 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. Several works present novel clock schemes for domino circuit design, such as methods for overcoming the non-inverting property of domino logic [1], and making the design more skew tolerant [7]. Other published research in this area includes methods for partitioning between domino and static gates [8] and for timing verification of domino logic [8], [9], [10], [11], [12].

In this paper, several stages of a domino synthesis and optimization flow, outlined in section 2, are discussed. In section 3, an efficient timing-driven static-domino partitioning algorithm is presented. The algorithm is then extended to two-way partitioning for domino logic and the applications of the two algorithms to various clocking strategies are discussed. In section 4, we describe a timing analysis and sizing optimization tool. Finally, concluding remarks are presented in Section 5.

II. Domino logic synthesis flow

Due to issues arising from the precharge and evaluation mechanism of domino logic and its non-inverting property, synthesizing domino logic is more complicated than standard two-stage static logic synthesis methods that perform technology-independent optimization, followed by technology mapping.

To build a high-performance domino circuit taking these factors into account, we suggest the synthesis flow of Figure 2 instead of a traditional library based synthesis flow. Step 1 corresponds to a standard technology-independent optimization. In Step 2, according to timing specifications and the area or power cost function, the input network is partitioned into the part implemented as static logic and the part implemented using domino gates. If domino gates are used, the choice of a clock scheme, and the partitioning of domino gates between clock phases is essential for correct and efficient performance of domino gates.

In Step 3, a technology mapping method that considers the characteristics of domino logic is applied. Due to the large number of candidate domino gate types, the use of a parameterized library and a library layout synthesizer is considered a good choice for this step [3], [4], [5].

While coarse timing optimization measures may be built

This work was supported in part by the Semiconductor Research Corporation under grant 99-TJ-692.

M. Zhao was with the Department of Electrical and Computer Engineering, University of Minnesota, Minneapolis, MN 55455, USA. She is now with Motorola SPS, Austin, TX 78729, USA

S. S. Sapatnekar is with the Department of Electrical and Computer Engineering, University of Minnesota, Minneapolis, MN 55455, USA.

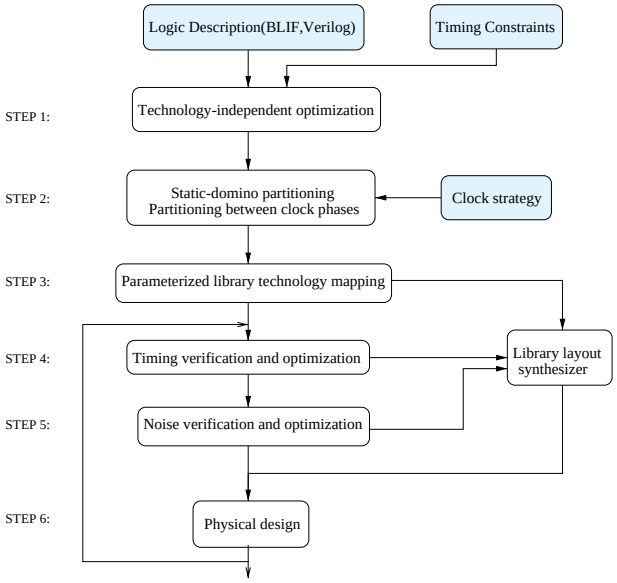


Fig. 2. Domino logic synthesis flow

into the preceding steps, there is a need for a rigorous timing verification step, followed by timing optimization to meet the set of applied constraints. This is performed in Step 4, where, for example, optimization methods such as transistor sizing are applied to meet long path constraint violations.

After these procedures, noise verification and optimization, performed in Step 5, is necessary. This is particularly important since domino circuits can be susceptible to noise effects such as charge sharing. This circuit is then passed on for physical design in Step 6, with iterations being performed until timing closure.

In this paper, we develop a static/domino partitioning algorithm for Step 2 of the flow, discuss briefly the technology mapping problem of Step 3, and then address issues related to Steps 4 of the flow by proposing a transistor sizing procedure.

III. Timing-driven partitioning of domino and mixed static-domino circuits

A. Introduction

After the technology-independent logic design step, a specified technology must be chosen to implement the input logic network. Both static and domino implementations offer various advantages. While static logic is commonly used for its robustness, its speed may be inadequate to satisfy the clocking constraints. Although various optimization methods such as sizing and buffer insertion can be used to increase the speed of the circuit, the area cost may increase exponentially beyond a certain point, and it is worthwhile to alter the logic style to meet such stringent constraints. Domino circuits typically provide higher speeds than static logic while having 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 use 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. This must be carried out while keeping the requirements of the specified clock scheme in mind.

In this section of the paper, we describe an automated design strategy to solve the following problems with the goal of minimizing an objective function such as area or power, under a set of timing constraints:

- We partition a circuit into static and domino regions, under the same clock phase. Note that since one of the partitions may be empty, the two extremes of constructing the circuit entirely using domino logic, or entirely using static logic, are also feasible solutions to the problem.
- Under a two-phase clocking scheme, we address the problem of partitioning the domino region to determine which gates are to be clocked by each clock phase.
- For a two-phase clocking discipline, we utilize the solutions for the above two problems to arrive at a flow that partitions a circuit into a common two-phase clocking strategy of the type illustrated in Figure 3.

As will be described through an example in Section III-B, for a partitioned circuit, the logic duplication penalty for the domino segment depends on the location of the cut between the partitions. Our method finds a partitioning solution that takes this into account while minimizing the an objective function.

To the best of our knowledge, the only published work addressing a related problem is [8], which outlines a static-domino partitioning procedure. Their work first implements the input logic using static gates, and then finds a critical path and its fanin transition region. A greedy approach is taken and the logic in this region is then remapped to domino gates to maximize the use of domino logic gates. No experimental results were presented in this work. In our paper, the problem is approached systematically, and a network flow based algorithm that provides an optimal solution, within the accuracy limitations of the cost estimation, is proposed. Our procedure is based on the following observations that differentiate it from [8]:

- Logic duplication can cause a large area penalty for large combinational circuits [3], [4]. 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.
- 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). Implementing this in domino logic as in [8] may be costly and unnecessary, and as long as the timing constraints are satisfied, it may not be essential to maximize the use of domino gates in the fanin cone.
- The work in [8] attempts to greedily minimize the number of domino transistors by utilizing them only in the critical region. We use an estimator that directly incorporates the area cost (including the duplication penalty) for static gates and domino gates to find optimal combinations between domino and static gates that minimize the overall

cost function, while meeting the performance targets.

B. Clock scheme and motivation

B.1 Clock scheme

Single-phase domino logic is impractical since precharging all domino gates simultaneously would result in wasted time during which no useful computation occurs. Therefore, domino circuits are typically partitioned into multiple phases and the design discipline imposes one of various allowable clocking strategies [1], [7], [8], [13], [14], [15]. For example, the use of two phases is common [13] where the precharge of one phase is overlapped with the evaluate of the other phase, as illustrated in Figure 3. The cycle boundaries in the figure may be latches active in Φ_1, or flip-flops launched at leading edge of Φ_2. If latches are used, then latches active in Φ_2 must be inserted at some position A, as illustrated in the figure. In the remainder of this paper, when we refer to a two-phase clocking scheme, we will denote the clock phases by the symbols Φ_1 and Φ_2.

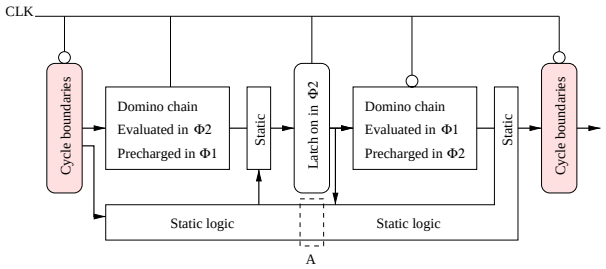


Fig. 3. A common clocking strategy for domino circuits

B.2 Problem definition and partitioning considerations

The problems of static-domino partitioning and domino partitioning between the two phases are critical problems to be solved in determining the optimal circuit implementation. We define these two problems concisely as follows:
Static-domino partitioning Given an optimized 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.
Two-way domino partitioning Given an optimized combinational circuit to be implemented entirely in domino logic and a two-phase non-overlapping clock scheme that permits inverters to be placed at the interface between the two phases, partition the boolean network into two clock phases such that the cost of domino implementation is minimized.
We will use the solutions to the above problems later in this paper to solve a more general statement of the partitioning problem in Section III-F.

For the problem of static-domino partitioning, we assume that we are provided with a combinational logic network description as an input, and that it can be partitioned into a static and a domino region within the same clock phase. Under this model, an additional precedence constraint that must be satisfied states that no static logic gate is permitted to fan out to a domino gate in the same clock phase when level-clocked latches are used within the static logic. A violation of this requirement would lead to functional errors in the domino logic [8]. The timing constraint for this situation is that the partitioned circuit must have its outputs ready at the end of the clock phase.

For the problem of two-way domino partitioning, we assume that the input is a combinational logic network that is to be partitioned so that its primary outputs are available in one clock cycle. This imposes a similar precedence constraint, where a domino gate in Φ_2 is not permitted to fan out to a domino gate in Φ_1. The two-phase non-overlapping clocking scheme forms a hard edge that prevents time borrowing between the phases. This results in a timing constraint that states that the delay within each phase must be restricted to a half-cycle.

The timing-driven partitioning method must consider two factors: the timing constraints, described above, and the cost of the implementation, measured in terms of the area or the power. In our experimental results, the area is modeled as the total number of transistors, but it is easy to incorporate a factor that models the clock routing penalty by adding a factor that is proportional to the number of domino gates.

An important consideration in the area cost relates to the requirement that only unate functions can be implemented in domino logic, as a consequence of the non-inverting nature of the domino logic family. Therefore, intervening inverters are handled by implementing a logic function and its negation separately, which duplicates the cost of logic implementation for that logic cone [6]. However, we observe that the duplication cost can be reduced by partitioning, and our formulation directly incorporates the cost of duplicating non-unate logic within a domino region in the cost function. To see this, consider the example shown in Figure 4.

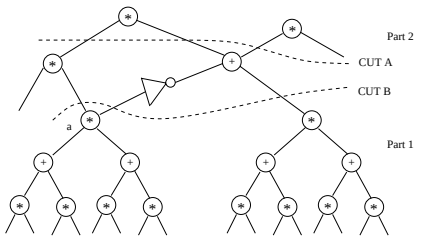


Fig. 4. An example for static-domino partitioning

In Figure 4, assume that the segment labeled "Part 1" corresponds to a domino implementation, while "Part 2" represents a static CMOS implementation. Two possible cut lines are shown in the figure. 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. Similarly, under the scenario where Part 1 corresponds to domino logic driven by phase Φ_1 of a two-phase clock, and Part 2

corresponds to phase Φ_2, with latches being placed at the cut, an identical argument holds, with the difference that the inverter for cut B would now be implemented using the Qbar output of the inserted latch.

C. Technology mapping for domino logic

Our static-domino partitioning algorithm requires the use of a technology mapper that estimates the cost of a static or a domino implementation for a region of the circuit. In this section, we will briefly describe the algorithm that is used in this work, though we emphasize that the partitioning method can be used with any other mapping algorithm.

Technology mapping is an optimization phase at the logic synthesis stage that binds the gates in the circuit network to the cells of a specific technology library. Although technology mapping for domino logic can follow the commonly-used static technology mapping procedure based on dynamic programming, special features of domino style gates such as large NMOS pull-down network, flexible gate configurations and logic duplication, demand further consideration. Compared to the mappers that simply migrate the algorithms for static logic, a procedure that makes use of these properties of domino logic will significantly improve the performance in both area and delay [16].

Various techniques for domino mapping have been introduced in the past [1], [3], [4]. The work in [3] applied the concept of a parameterized library for domino mapping, and our mapper uses this same idea. In contrast with a conventional library that consists of a small set of precharacterized cells, 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. Particularly in a domino environment, where pulldown networks typically have a large width and height, the number of possible gates in a parameterized library can be extremely large [17]. Therefore, it is infeasible to pre-layout and precharacterize all of the large number of possible configurations that belong to the library, and after the circuit is technology mapped on to the parameterized library, the layout of the required library cells must be produced by an on-the-fly cell generator. The greater flexibility afforded by such a technique in selecting cells from this virtual parameterized library can overcome the deficiencies associated with the above limitation [18].

In an environment that uses a parameterized library, we state 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. A fast structural matching technique based on dynamic programming [16] is applied for fast parameterized library mapping. We explore the design space by storing the sub-solutions for all possible [height, width] combinations from $[1, 1]$ to $[H, W]$, where H is the maximal height of domino gates and W is the maximal width of domino gates. Each sub-solution is optimal for its corresponding substructures. The basic structural mapping algorithm is further augmented by techniques that incorporate the use of domino-specific circuit structures such as dual-monotonic domino gates, dynamic-static gates [19], and multiple-output gates into the technology mapping framework.

D. Timing-driven static-domino partitioning

D.1 Algorithm outline

The two subproblems listed in Section III-B.2 can be solved using similar algorithms, using different cost formulations. In this section, we will illustrate the algorithm for static-domino partitioning. The algorithm applies a DAG technology mapper based on dynamic programming, a timing analysis method based on PERT, and a maximum flow algorithm to realize a timing-driven two-way partitioning.

The input to the algorithm is an arbitrarily optimized two-input AND-OR DAG network. In outline, the algorithm consists of the following steps that are described in detail in the remainder of this subsection:

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

B. 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.

C. Build the flow network from the candidate cut nodes, assign edge capacities, and apply a maximum flow algorithm to the network to obtain a minimal cost cut [20].

D.2 Step A: Cost estimation

The first step of partitioning is to perform static and domino mapping on the logic network to obtain cost estimates. The partitioning between domino and static requires a cost comparison of each vertex of the network implemented using a domino or a static implementation. While we would like to perform this determination on the same network for purposes of storage efficiency, this task is complicated by the fact that technology mapping of domino logic requires the input network to be unate; after unating, the topology of the new network will be different from the original network on which static mapping is performed. Moreover, inverters may be pushed through the network using De Morgan's laws, and therefore their location is not fixed. Depending on where the inverters are placed, the cost of logic duplication and the positions at which logic duplication occurs could vary, and a good partitioning algorithm should consider all such possibilities in arriving at a partition.

To address these issues, we introduce the idea of a *twin network*, N_{twin}, that represents the original network. Each vertex in N_{twin} corresponds to a node in the original network, and stores information on the implementation of both 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 the N_{twin} is negative; otherwise, it is positive. It is not necessary to store inverters in the original network as nodes in the twin network; rather, these may be merged into the polarity of an edge as shown in Figure 5, where inverter node d is collapsed into the edge between a and b by setting its polarity to be negative. The lists of positive and negative polarity fanouts of each vertex in N_{twin} are maintained, and in case of domino mapping, the duplication of any fanout is flagged.

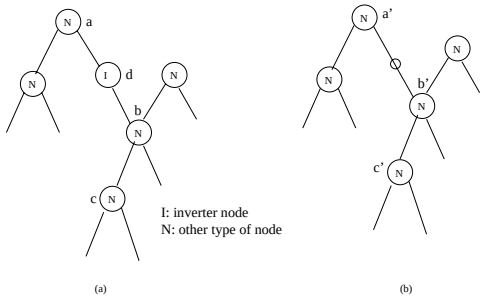


Fig. 5. Inverter elimination from the original network (a) in the twin network (b)

Once the technology mapping results on N_{twin} are available for both for static and domino implementations, cost estimations are available for both the static and domino implementation at each vertex.

Definition 1: We define the following quantities at each node N.

$p\_delay_d(N)[n\_delay_d(N)]$:
the positive [negative] polarity delay from the inputs to node N using a domino implementation.

$p\_delay_s(N)[n\_delay_s(N)]$:
the positive [negative] polarity delay from the inputs to node N using a static implementation.

$p\_area_d(N)[n\_area_d(N)]$:
the area cost of implementing the positive [negative] polarity logic of node N using domino gates.

$p\_area_s(N)[n\_area_s(N)]$:
the area cost of implementing the positive [negative] polarity logic of node N using static gates.

The cost estimation at each vertex includes both delay and area information, and the above quantities are calculated at each node. Note that $p\_area_s(N)$ and $n\_area_s(N)$ can differ by at most two since one can be obtained by placing an inverter at the output of an implementation of the other. Therefore, storing only one of the two values will lead to a negligible loss in accuracy. However, for domino circuits, due to the requirement of the unateness property, the positive and negative polarities of a node are implemented separately, which may lead to a significant difference in their values. We heuristically use $delay_d(N) = \min(p\_delay_d(N), n\_delay_d(N))$ as the largest delay from the inputs to node N using a domino implementation and $delay_s(N) = \min(p\_delay_s(N), n\_delay_s(N))$ as the largest delay from the inputs to node N using a domino implementation.

D.3 Step B: Determining the candidate cut nodes

The objective of the first step of partitioning is to obtain a coarse cost estimate for delay and area of same network for both static and domino implementations. Once this cost estimation has been obtained for each vertex, the next task is to determine the region that can be partitioned without violating the timing constraints. We will now introduce the idea of a *candidate cut node*, illustrated in Figure 6.

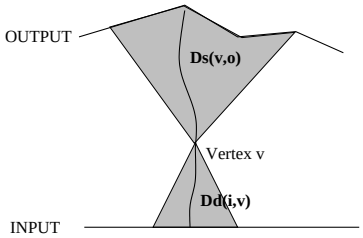


Fig. 6. Determination of candidate cut nodes

Assume that v is some vertex in N_{twin}. If the cutset passes through the node, then the input transition cone of v 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 feasible; if not, it is certain to violate the timing constraint. The procedure of finding candidate cut nodes makes this determination at each vertex.

The value of $D_d(i,v)$ is directly obtained from $delay_d(v)$ from the technology mapping phase. The value of $D_s(v,o)$ can be obtained using a PERT-like procedure [21] to traverse from the outputs of the network towards its inputs. The algorithm for finding candidate cut nodes is as follows:

ALGORITHM: **Find_candidate_cut_nodes**
For all nodes i, $D_s(i,o) = 0$;
Perform a PERT traversal from outputs to inputs.
At each node v
{
If $D_s(v,o) + delay_d(v) \leq T_{spec}$
v is a *candidate cut node*;
node_delay = $\min[delay_s(v) - delay_s(i)]$, $\forall i \in inputs(v)$;
For each input i of v
$D_s(i,o) = \max[D_s(i,o), \text{node\_delay} + D_s(v,o)]$;
}

D.4 Step C: Finding the minimum cut

The objective of Step B is to find the candidate cut nodes of input network that satisfy the imposed timing constraints. After all of the candidate cut nodes have been found, a maximum flow network, whose minimum cut corresponds to minimal cost of mixed domino-static imple-

mentation, is built. The max-flow min-cut algorithm is then applied to this network to find the minimum cut. This procedure consists of the following steps:

I. Maximal flow capacity assignment

The DAG technology mapping procedure performs the mapping of a Boolean network to the gates in a parameterized library. For a multi-fanout node in the network, the area contribution of the node is divided by the fanout count of the node, as in [22], so that the area value of each node, $area(N)$, is the approximate 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 each cut set node can be approximated as $\sum_{i \in CutSet} area(N_i)$, where the value of $area_d(N_i)$ is set to $\min[p\_area_d(N), n\_area_d(N)]$. In the example of Figure 7, if nodes a, b, c and d form the cut vertices, then the area of region Y is calculated as $area_d(a) + area_d(b) + area_d(c) + area_d(d)$.

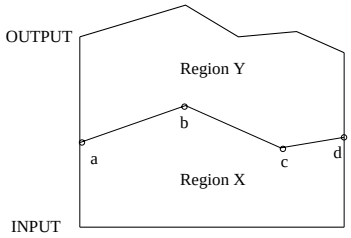


Fig. 7. Evaluating the cost of a cut

For a given cut that divides the network into two parts, referred to as Region X (closer to the inputs) and Region Y (closer to the outputs), the cost can be calculated as follows. Assume that the area cost of the entire network implemented in static logic is $A_s(sum)$, and the area 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 $A_s(sum) - A_s(X) + A_d(X)$. Since $A_s(sum)$ is constant over all partitions, the objective of minimizing cost of mixed implementation 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 a network where the capacity associated with each vertex i is set to $area_d(i) - area_s(i)$.

II. Building the maximum flow graph

The maximum flow algorithm is a well-established approach for finding minimum cost cuts. The procedure of building the maximum flow network can be illustrated on the example circuit of Figure 4.

After performing static and domino mapping on N_{twin} and determining the candidate cut nodes, we obtain an example circuit shown in Figure 8(a). The shaded part of the network shows the region containing the candidate cut nodes. There are two weights on every node, which represent the result of domino mapping, $area_d(i)$ and the result of static mapping, $area_s(i)$, respectively.

Finding the minimum cost cut on the above network is equivalent to finding the minimum cost cut on the maximum flow graph shown in Figure 8(b). In Figure 8(b), the nodes in the shaded region that are closest to the primary inputs of Figure 8(a) are connected to the source node, and the nodes of the region that are closest to the primary outputs in the original network are connected to the sink node. Each vertex in Figure 8(a) is split into two vertices connected by an edge of capacity $area_d(i) - area_s(i)$.

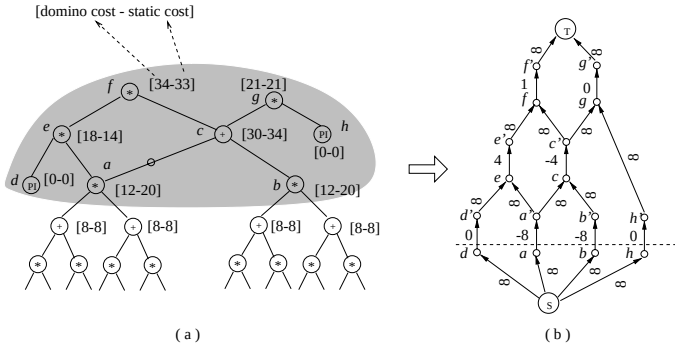


Fig. 8. Boolean network after mapping and candidate cut node decision

However, before the standard max-flow min-cut 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 node, unless that logic is related to the output inversion function of a domino gate. A solution to this problem is provided in [23], [24].
2. Standard maximum flow algorithms cannot handle edges with negative capacities, and the network must be modified suitably.

To solve the problem, we heuristically transform the vertex-cut maximum flow network into an edge-cut maximum flow network and then translate it into a standard maximum flow network with nonnegative edge capacities. The procedure consists of the following steps:

1. Building the edge cut maximum flow network.
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 $C_{init} = (area_d(u) - area_s(u))/fanout(u)$, where $fanout(u)$ is fanout number of node 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 $Flow(u, v) = (\sum_{i \in input(u)} Flow(i))/fanout(u)$. Since this is a feasible flow, updating the capacity C of each edge as $C_{modified}(uv) = C_{init}(uv) - Flow(uv)$ leaves the identity of the minimum cost cut unchanged. This procedure is repeated until the value of $C_{modified}$ for each edge is nonnegative; it is easily verified that this method must converge.
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 three steps described above are illustrated, respectively, by Figures 9(a), 9(b), and 9(c). The max-flow min-

cut algorithm is then applied to the network to obtain the minimum cost cut.

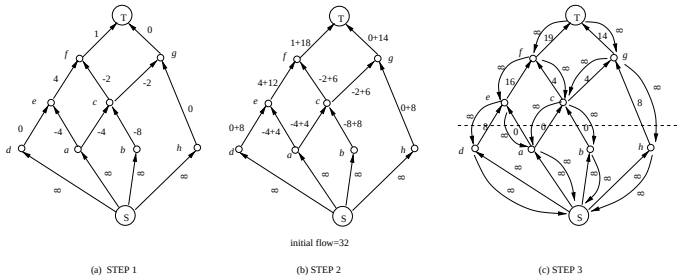


Fig. 9. Constructing the edge-cut maximum flow network

E. Two-way domino partitioning

As described in Section III-B.1, even a circuit that is implemented purely in domino logic is typically operated in two phases, so that the precharge of one phase overlaps with the evaluate of the next phase, thereby ensuring that useful computations are carried out in the circuit at all times. In this section, we address the problem of two-way domino partitioning of a circuit under a two-phase non-overlapping clock. The objective here is to determine which gates in the circuit should be clocked by the first clock phase, and which by the second clock phase, with inverters permitted between the two phases. Like the static-domino partitioning scenario described earlier, there is an inherent precedence constraint that states that no gate clocked by Φ_2 may fan into a gate clocked by Φ_1. Another feature common to the two problems is that the partitioning itself influences the cost of implementation area. The differences between the solution methods lie in the manner in which the candidate cut node is chosen, and how the capacities in the max-flow network are assigned, and are summarized as follows:

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, since candidate cut nodes are those that satisfy the timing constraints in each phase and no time borrowing between two phases, 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.

Maximal flow capacity The two clock phase regions must be separated by a latch, which provides the ability to invert a signal. The cost function here is the cost of latches and the cost of logic duplication.

In the input network, some of the logic nodes must be duplicated since the logic function is required in both true and complemented forms, while some of the other logic outputs are required only in one polarity and need no duplication. There are two possible cases:

• Assume that N is a node at which both the true and complemented form of the logic function are required. If this node does not belong to the cutset, then the logic must be duplicated, and hence the area cost at this node is $p\_area_d(N) + n\_area_d(N)$. If the node lies on the cut, then only one polarity of the logic needs to be generated and the other is generated by an inverter or within the latch placed at the cut. Therefore, the area contribution of the node can be modeled as $\min[p\_area_d(N), n\_area_d(N)]$. Therefore, the area cost difference between placing node N on the cutset or not is the $\max[p\_area_d(N), n\_area_d(N)]$, plus the latch area.

• On the other hand, for a node N at whose output only one polarity is required, the area cost difference between passing a cut through N or not is merely the area of the latch.

In either case, the capacity corresponding to a cut at this node is represented by the area cost difference values defined above.

The two-phase non-overlapping clock creates a hard edge at the latches that prevents time borrowing between the two phases. On the other hand, an overlapping two-phase clocking scheme is often used [7] to enable cycle borrowing. In such a case, the step of determining the candidate cut node is a simple extension of the procedure described earlier. As before, 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; however, it should be recalled that in an overlapping two-phase clocking scheme, the clock pulse width is more than half of clock period. It should be noted that when time borrowing is used, the existence of an inverter between the two-phases could violate the timing constraints of domino logic. In this case, the cost of the inserted latch instead of the cost of reduced duplication allowed by an inserted inverter would become the main factor to be considered and should used to assign the maximal flow capacity.

F. A partitioning flow for a general two-phase clocking strategy

A flow for partitioning an input network into stages of the general two-phase clocking scheme illustrated in Figure 3 is more complex than the procedures described earlier in this paper. We solve the problem heuristically by applying a design flow based on the timing-driven static-domino partitioning algorithm of Section III-D and the two-way domino partitioning algorithm of Section III-E. If flip-flops are used at the cycle boundaries, **Flow 1** below is applied:

1. We first perform static-domino partitioning on the entire network to divide the circuit into domino and static regions, labeled, respectively, as Region 1 and Region 2. The timing constraint specified 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 on this region to obtain the domino partition to be (tentatively) clocked by phase Φ_1 of the clock (Region 3) and the region to be clocked by phase Φ_2 of the clock (Region 4).
3. Region 3 is now assigned to Phase 1; however, we now investigate the possibility of implementing a part of it in static logic. Therefore, we perform static-domino parti-

tioning on this region to divide it into a phase Φ_1 domino region (Region 5) and a phase Φ_1 static region (Region 6). The end 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.

We propose another possible partitioning design flow, referred to as **Flow 2**. It first performs two-way domino partitioning on the input circuit, followed by a static-domino partitioning step on each of the two resulting partitions. This flow is more appropriate for use in a situation where level clocked latches are to be used, since, unlike **Flow 1**, it requires the use of a level-clocked latch even within the static logic (as shown by latches in position A in Figure 3).

G. Cost modeling

In the discussion so far, we have used the area as the cost function for partitioning. However, the algorithm may be adapted to other cost objectives, using a power model, or an area model with an incorporated routing cost. A transistor level Elmore delay model that is similar to that in [25] has been used in this work. While the Elmore delay is known to be limited in its accuracy, it is reasonable to use it here for a fast and approximate delay estimate, since the partitioning procedure is carried out early in the design flow shown in Figure 2.

The delay of a domino gate is given by

$$D_{domino} = R_n \cdot (1+S) \cdot C_{gate} \cdot (1 + cpnratio) + Delay_{para} + R_p \cdot C_{gate} \cdot N_{fanout}, \quad (1)$$

where R_n, R_p are the driving resistances of the NMOS transistor and the PMOS transistor, respectively, S is the maximum height of the NMOS pull-down network, C_{gate} is the gate capacitance for a unit NMOS transistor, and *cpnratio* is defined such that the gate capacitance of the PMOS transistor on the output inverter is $cpnratio \cdot C_{gate}$. 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 effects of the internal capacitances are incorporated by the second term, $Delay_{para}$, using the Elmore delay metric, and the last term represents the delay of the inverter that is driving the fanout gate capacitance.

Separate rising and falling delays were used for static gates, given by the equations

$$D_{r,static} = R_n \cdot S \cdot C_{gate} \cdot ((1 + cpnratio) \cdot N_{fanout}) + Delay_{para}$$

$$D_{f,static} = R_p \cdot P \cdot C_{gate} \cdot ((1 + cpnratio) \cdot N_{fanout}) + Delay_{para},$$

where S and P are, respectively, the maximum height of the NMOS segment and the PMOS segment of the static gate, and the other terms are analogous to those defined in the previous paragraph.

H. Experimental results

The entire partitioning and technology mapping flow has been implemented using C++ and applied to the multi-level circuits in the LGSynth91 benchmark suite. A parameterized library based mapping algorithm that is similar to [5] is used for the domino technology mapping. The parameterized library contains all possible cells with up to 4 transistors in series and 4 transistors in parallel. A method using the same principles as [5] is used for the static technology mapping to a parameterized library that contains all possible cells with up to 3 transistors in series and 3 transistors in parallel. The mapping objective is area minimization, where the area is measured in terms of the number of transistors. An initial technology-independent logic minimization for all circuits is performed using SIS with *script.rugged* script file, after which the input networks are compressed to form the N_{twin} network addressed in Section III-D.2. The delay estimation is carried out using the delay model described in Section III-G. The primary inputs are assumed to be available in true and complemented form, and it is assumed that each of these is driven by a flip-flop that provides the inverted signal at no cost.

The results of static-domino partitioning are shown in Table I. Columns 2 and 3 show the implementation cost of pure one-phase domino mapping and pure static mapping, respectively. The delay of the static implementation is also shown. All delays in this table are normalized so that "×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. In particular, for the timing-unconstrained case, the delays of the optimal mixed static-domino implementation are listed in Column 6. The average CPU execution time of the partitioner, including technology mapping, is shown for each circuit in column 10.

From the table, we observe that domino implementations of the benchmarks typically 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. No special fanout optimizations were performed on our benchmark. With fanout optimization, greater delay improvement are expected for static logic than domino logic. Under loose timing constraints, for most circuits, the cost of the mixed static-domino implementation is no larger than the minimum of pure static and pure domino implementation costs; this is as expected since they form two feasible solutions to the partitioning problem. The exceptions to this are the benchmarks "count" and "c2670," which have a slightly larger area than the static implementation. Several factors lead to this minor inconsistency in the partitioning algorithm. Firstly, during the cost estimation, we heuristically estimate the cost of each node by dividing the cost of multiple fanout nodes by the number of fanouts. Secondly, for domino implementations, the number of fanouts itself depends on whether the fanout region of a node will be duplicated or not, and this is not considered by the algorithm. Thirdly, the approximations used in maxflow capacity assignment as described in Section III-D.4 are partially responsible for this difference. We point out, however, that

TABLE I
RESULTS OF THE STATIC-DOMINO PARTITIONING ALGORITHM

Circuits	Domino	Static	No spec			Spec=(×1.25)		Spec=(×1.05)		CPU
	N_{tran}	N_{tran}/delay	N_{tran}	G_d/G_s	delay	N_{tran}	G_d/G_s	N_{tran}	G_d/G_s	(s)
c1355	1824	1302/2.25	1302	0/260	2.25	1800	170/104	1800	170/104	1.4
dalu	2360	2192/2.16	2098	97/198	1.78	2096	147/132	2096	189/75	7.9
c880	1163	982/2.08	958	21/124	2.07	1015	56/88	1027	62/85	1.4
count	357	336/2.77	344	5/54	1.71	350	23/30	353	32/18	0.3
c1908	1978	1308/1.78	1306	5/263	1.78	1723	174/86	1928	238/34	1.4
c2670	1992	1754/1.43	1775	79/173	1.15	1775	79/173	1774	81/170	3.5
c3540	4527	2850/1.78	2748	88/349	1.78	3312	260/218	3987	461/26	10.9
c6288	13702	8350/1.81	8340	16/1771	1.76	12079	1301/493	13456	1733/73	33.5
k2	2884	2896/1.54	2884	368/68	0.95	2884	368/68	2884	368/68	8.6
des	9945	8134/4.25	7527	160/915	1.00	7536	165/911	7536	165/911	60.2
c7552	7919	5464/2.35	5396	78/857	2.35	5987	375/578	6198	456/504	30.9
t481	1697	1832/1.35	1695	203/19	0.97	1695	203/19	1695	206/15	3.7
rot	1777	1536/1.99	1462	55/171	1.99	1514	87/137	1611	126/103	3.0

in all cases, the error is small enough (2.4% for count and 1.2% for c2670) that the above drawbacks are outweighed by the massive reductions in CPU times that they permit.

The results of the two-way domino partitioning algorithm are shown in Table II. Column 2 shows the results of one-phase domino implementation and is identical to Column 2 of Table I. Columns 3-5 show the two way partitioning results without any timing specification, while columns 6-7 and 8-9 show the results under timing specification of $\times 1.25$ and $\times 1.05$, respectively. An explanation of the above timing specifications is as follows: if the maximum delay for the domino mapping is Δ, then $\times 1.05$ specification represents the fact that a delay constraint of $\Delta \times 1.05 \times 0.5$ is assigned to each phase of the domino partition, corresponding to the fact that the total delay must be evenly distributed over the two partitions for a symmetric nonoverlapping clock scheme. The results under the specified time constraints include the area cost of domino logic, the number of inverters and the number of latches. The delays for a partitioning solution that minimizes the area under no timing constraints are listed in column 5. In most cases, the resulting area is no smaller than that for the tighter specification, as expected. As in static-domino partitioning, in a few cases, a slightly smaller area is obtained (but within a reasonable margin of error) due to approximations in the partitioning algorithm. The differential between the numbers in columns 2 and 3 shows the reduction in logic duplication obtained by exploiting the inverters at the partition boundary, and this is often significant. The CPU execution time for the two-way domino partitioning procedure is shown in the last column and can be seen to be very reasonable.

The results in Tables I and II are applicable to a single-phase static-domino implementation and a two-way domino implementation, respectively. For the general two-phase clocking scheme shown in Table III, we apply the techniques listed as **Flow 1** and **Flow 2** in Section III-F. For each circuit, we list the number of transistors, N_{tran}, and the number of latches, N_{latch}. The experiments are executed under two sets of delay specifications, namely, $\times 1.05$ and $\times 1.25$, where all times are normalized to a pure domino two-phase implementation. From the results, we can see that the results of using a combination of static and domino

TABLE III
RESULTS OF APPLYING THE PARTITIONING FLOWS FOR THE TWO-PHASE CLOCKING SCHEME

Circuits	N_{tran}/N_{latch}			
	Flow 1 ×1.25	**Flow 1** ×1.05	**Flow 2** ×1.25	**Flow 2** ×1.05
c1355	1408/8	1456/8	1452/48	1486/48
dalu	1998/56	2050/78	1848/122	1966/94
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	1538/52	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
t481	1701/84	1752/90	1831/80	1833/85
rot	1463/36	1515/51	1485/118	1538/119

gates yields areas that are similar to those in static implementations, and speeds that are similar to domino speeds, thereby providing the best of both worlds. As the timing constraints are made more relaxed, the resulting area can become even smaller, barring minor anomalies due to our heuristic approximations, whose effects are seen in the circuit "count" under **Flow 1**. Therefore, our method can be used to find an area-delay tradeoff curve for static-domino partitioning.

It is observed that **Flow 2** introduces a larger number of latches than **Flow 1**. This can be explained by the fact that the two flows are directed towards different clocking disciplines: in **Flow 1**, the cycle boundaries are edge-triggered flip-flops, while in **Flow 2**, they correspond to latches. Moreover, **Flow 2** begins with an initial two-way domino partition and hence must insert latches at the boundaries, even if they are not essential to the final partition. It is generally true that in Table III, the results of **Flow 1** are better than the results in Table I, while the results of **Flow 2** are usually better than those in Table II. This is a natural consequence of how these flows are created, based on combining the static-domino partitioning and two-way domino partitioning algorithms; however, since the method is partly heuristic, a couple of the circuits do not obey this property.

TABLE II
RESULTS OF THE TWO-WAY DOMINO PARTITIONING ALGORITHM

Circuits	Domino	No spec			Spec=(×1.25)		Spec=(×1.05)		CPU
	N_{tran}	N_{tran}/N_{inv}	N_{latch}	delay	N_{tran}/N_{inv}	N_{latch}	N_{tran}/N_{inv}	N_{latch}	(s)
c1355	1824	1656/32	32	1.59	1600/16	16	1600/16	16	1.8
dalu	2360	1881/73	122	0.90	1881/73	122	1846/74	94	10.0
c880	1163	933/22	64	1.24	1025/17	38	1037/14	43	4.6
count	357	267/15	50	1.09	316/7	24	347/7	23	0.4
c1908	1978	1861/37	67	1.32	1733/45	72	1838/40	60	2.7
c2670	1992	1703/74	106	0.95	1703/74	106	1703/74	106	6.0
c3540	4527	3499/44	50	1.69	3418/86	51	3407/94	53	11.7
c6288	13702	13173/34	39	1.47	13114/58	58	13085/108	108	96.4
k2	2884	2856/1	46	2.00	2866/1	159	2921/1	147	9.6
des	9945	8265/37	293	1.66	10835/5	437	10835/5	437	111.5
c7552	7919	6413/201	251	1.21	6413/201	251	6607/186	225	44.6
t481	1697	1632/14	31	1.80	1744/0	72	1823/0	80	5.43
rot	1777	1422/53	152	1.03	1572/41	131	1638/46	118	4.9

In all the above tables, we compare all partitioning results with pure one-phase domino. The area advantage of partitioning is more than just the ratio of partitioning circuits cost to pure domino implementation. Domino circuits are rarely implemented as pure one-phase domino. They are partitioned into several phases, and the possible area overheads include splitting gates, the use of latches between stages, buffer insertion if there is no logic in a given stage, and the cost of clock generation and routing.

To summarize the results, we reiterate that the objective of the algorithms is to minimize the implementation cost while satisfy the timing constraints, under various clocking schemes. The cost improvements arise from two quarters: from the freedom to select either domino or static implementations in parts that are not timing-critical, and from the reduction of logic duplication for domino logic due to the partitioning scheme. Although our cost model does not explicitly consider the cost of clock generation and routing, which is a drawback of domino logic, this may easily be factored in as being proportional to the number of domino gates.

Our implementation ignores second-order effects such as the effects of clock skew, and the penalty due to the presence of latches between the two phases. A latch between the two phases may be implemented as a regular latch with 6-8 transistors, or by adding an extra transistor to the output inverter of the preceding domino gate, as in Figure 14 of [7]. Adding these costs to the algorithm is relatively easy, since it requires the cost of a latch to be added to the cost of each edge of the vertex cut graph, so that the min-cut counts the cost of one latch for each edge in the cut.

IV. TIMING VERIFICATION AND SIZING OPTIMIZATION

A. Introduction

The correct functionality of a circuit containing domino logic is contingent on correct timing relations between the clock and individual domino gates. For a given circuit topology consisting of static and domino gates, timing verification and optimization is an essential step of the design process. In different designs, domino circuits may utilize one of various clock schemes, such as the clock strategy of Figure 3, skew-tolerant overlapping clock [7], self-timed domino [13], wave-domino pipeline [14], and clock-delayed domino[1]. The goal of the section is to provide a general timing verification and sizing tool for mixed static-domino circuits that presents the problem in a similar framework as the corresponding solutions for static circuits.

Previous research in sizing domino circuits has been limited. Several sizing algorithms have been published in the past (a survey is provided in [21]), but most of them have not considered domino logic. Although the research of [11], [12] performs sizing for domino circuits, both techniques perform local optimizations, optimizing only one domino block at a time. In [26], a sizing tool for domino style circuits called Focus was developed.

B. Domino logic timing constraints

In this section, we summarize the timing constraints associated with domino logic. For details, the reader is referred to the timing analysis techniques in [9], [10]. A less conservative timing constraint is described in [8], but is not incorporated into this work.

Let us first consider the timing constraints of an individual domino gate used in our timing analysis and sizing optimization tool. The clock input to the domino gate is shown in Figure 1. The precharge phase begins at $T_{clk,f}$ and continues until $T_{clk,r}$, and the evaluate phase begins at that time and ends at time $T_{clk,f} + P$ where P is the period of the clock signal feeding the domino gate. The reference time $t = 0$ is set with respect to the clock signal at the primary input of the circuit block. If more than one clock signal is used, any one of them may be used as a reference, and the transition times of the other clocks may be expressed according to the reference.

The long path timing constraints we used for each domino gate include:
(1) The falling events at data input nodes should meet the setup-time requirement to the rising edge of the evaluate clock.
(2) The rising event at the output node must be completed before the falling edge of evaluate clock.
(3) The rising event at dynamic node d of the domino gate must be completed before the rising edge of the evaluation clock. This constraint implies that the pulse width of

precharge clock must be capable of pulling up the output node.

For the short path timing constraints, instead of using the conservative constraints of [9] or more aggressive constraints of [10], we use different constraints for falling and rising edges. These constraints, which are more accurate and capture the requirements for the correct functioning of domino logic, are listed below:
(1) For any rising input data line, a transition should occur only after the end of evaluation stage.
(2) A falling input data line may not go down to logic 0 but must be held at the logic 1 level until the output transition in that cycle has been completed. It can be given by

$$t_f(in) + P \geq T_r(out) \tag{2}$$

where $t_f(in)$ are the earliest falling event time of input signals, $T_r(out)$ refers to the latest rising event time at the output node, and P is the period of the clock signal feeding the domino gate.

Note that if the falling signal arrives before the end of the evaluation phase, it will not influence the output as long as the dynamic node has already been discharged. If, however, the rising signal arrives earlier than the end of the evaluation phase, it may cause a glitch at the output. The presence of $T_r(out)$ in Equation (2) causes the short path timing analysis to be related to the long path timing analysis.

C. Timing verification

The timing analysis procedure described here is based on the PERT procedure and uses a table-lookup delay model for delay calculation. The long path timing analysis consists of two steps.

First, the circuit is forward-traversed, beginning with the primary input nodes and the clock node. The rising and falling event arrival times for each node v are calculated as follows:

$$T_r(v) = \max(T_f(u) + D_r(u,v)) \tag{3}$$
$$T_f(v) = \max(T_r(u) + D_f(u,v) \tag{4}$$

where $T_r(u)$ and $T_f(u)$ are, respectively, the rising and falling event times for nodes u, and $D_f(u,v)$ and $D_r(u,v)$ are, respectively, the worst fall delay and rise delay from input u to output v. The domino clock input node is treated in the same way as any primary input node, and the rising or the falling edge of the clock provide the corresponding event times for the clock node. The rising and falling event arrival times at the output node of a domino gate can be obtained similarly to the static gate arrival time computations, using equations (3) and (4). The only difference is that the rising event at the dynamic node is related only to the falling edge of the domino clock and is independent of the other input nodes. This fact is captured by setting the value of D_r from each input node of the domino gate to the output node as $-\infty$. At the end of this traversal, the arrival time at each node has been computed.

Next, a backward traversal is carried out to calculate the required time at each nodes. Before traversal, the required time of domino gates from long path constraints in Section IV-B and the required time at each primary output from synchronizer have been assigned the corresponding nodes. Beginning with the primary output nodes, we make a reverse PERT pass back through constraint graph to compute the required time at each node and the slack associated with every edge. At the same time, the critical path, defined as the path with maximum negative slack, is found. During this second traversal, the algorithm keeps a record of the minimum negative slack. If an edge with the same slack is encountered, then it is incorporated into the critical path. If an edge with smaller (more negative) slack is traversed, then the previous critical path is discarded and the critical path is updated to begin at that edge. Note that due to the presence of constraints at each domino gate, the critical path does not necessarily terminate to a primary output and could terminate at a domino gate instead. Additionally, it is possible for the origin of the critical path to be either at a PI or at the clock node.

After long path timing analysis has been performed, $T_r(out)$ in constraint (2) is available. A PERT-based procedure that is similar to long path timing verification can be applied to verify the two short path constraints addressed in IV-B.

D. Sizing algorithm

If long path timing constraints are found to be violated after applying the above timing analysis procedure, a sizing algorithm is applied. In addition, the short path timing constraint (2) is incorporated into the sizing procedure. Suppose that $T_r(i)$ and $T_f(i)$ are, respectively, the rising and falling event times for arbitrary node i, the sizing problem is formally stated as follows:

$$\begin{aligned}
&\text{minimize} && Area \\
&\text{subject to} && \\
&\max(T_r(o), T_f(o)) \leq T_{spec} && \forall o \in PO \\
&T_f(in) \leq T_{clk,r} - T_{setup} && \forall in \in I_{domino} \\
&T_r(out) \leq T_{clk,f} + P && \forall out \in O_{domino} \\
&T_r(d) \leq T_{clk,r} && \forall d \in D_{domino} \\
&T_r(out) \leq t_f(in) + P && \forall out \in O_{domino} \\
&K_1 \leq \frac{W_p}{W_n} \leq K_2 && \forall \text{ gates in the circuit.}
\end{aligned} \tag{5}$$

where *Area* is the area of the circuit and, as in other work on transistor sizing [25], is approximated as a sum of transistor sizes. The first set of constraints represents the constraints from synchronizer, where PO is the set of primary outputs and T_{spec} is the time specification for primary outputs. The second through the fourth set of constraints correspond to long path constraints of each domino gate, where i_{domino}, o_{domino} and d_{domino} are, respectively, the set of inputs, outputs dynamic nodes of all of the domino gates in the circuit, and T_{setup} is a constant that acts as

a safety margin. The fifth line of constraints come from short path constraints 2, and the last set of constraints are related to the noise margin, where $W_p(W_n)$ refers to the width of the p-(n-) transistor in the equivalent inverter associated with each complex gate.

The sizing algorithm used here is an adaptation of the TILOS algorithm [25]. Beginning with a circuit where all transistors are minimum-sized, each iteration selects one transistor and increases its size by a constant factor. In each iteration, a timing analysis is performed to identify the constraint $g(\mathbf{w}) \leq 0$ with the largest violation, where $g(\mathbf{w})$ denotes the fact that the constraint g is a function of the vector $\mathbf{w}$ of transistor widths. The traceback procedure described above is used to determine the critical path of the circuit, which corresponds to that constraint. The sensitivity of the constraint function g to each transistor width is computed, and the width of the transistor with the most negative sensitivity is increased. The iterations continue until the timing specifications are all met, or until no further improvement is possible.

E. Experimental results

The timing verification and sizing tool has been implemented in C++, and takes an input in the format of a SPICE transistor netlist. The characterization was performed in a 0.5 μm technology. The bump-size of each sizing step for the TILOS-like algorithm is set to $\times 1.5$ of original size. The noise margin constraints require that $K1 = 1.0$ and $K2 = 4.0$. The procedure has been performed on two sets of circuits of different clock strategies. A summary of the results are listed in Table IV and Table V, respectively. For each circuit, the original unsized area, which is obtained by setting each transistor minimal size, is listed. For various domino clock specifications listed in the "Clk spec" column, the results of sizing are listed. The output timing specification is always set to the corresponding clock period, i.e., $T_{spec} = T_{clk,f} + P$. The area is reported as "-" if the specifications are too tight to be satisfied.

TABLE IV

Transistor sizing on circuits for the two-phase clocking scheme

Circuit	Unsized Area	Clk spec period(s)	Optimized area	CPU (s)
C1355	1582	1.8	1589	1.2
		1.6	1605	1.5
		1.4	-	2.3
dalu	2357	2.3	2453	7.2
		2.08	2524	10.4
		1.86	2711	15.7
		1.64	-	17.0
C1908	1764	2.6	1780	2.1
		2.2	1871	6.3
		1.8	2107	16.5
		1.4	-	31.8
des	9881	4	9910	10.1
		3.6	9977	13.7
		3.2	10156	26.4
		2.8	10506	57.1
C7552	6948	3.5	6954	5.9
		3.0	7015	14.8
		2.6	7210	38.7
		2.1	-	73.5

The first set of results, reported in Table IV uses the two-phase clocking scheme of Figure 3, whose duty cycle is set of 1/2. We also apply our procedure to four-phase overlapping domino clocks of [7] with a duty cycle of 1/2, illustrated in Figure 10, and list the results in Table V. Note that for the same set of circuits, the unsized area is different for the overlapping and nonoverlapping schemes: the nonoverlapping clock scheme does not permit inversions at internal latches that use time borrowing, and therefore requires more logic duplication. Each block consists of several levels of domino gates, or clocked buffers if no domino gates are available. Both the long path and short path timing verification are performed for the input circuit in each case. It is easily verified that as the clock specification is made more stringent, the area of the circuit increases in each case.

TABLE V

Transistor sizing on circuits for a four phase overlapping clock with a 66% duty cycle

Circuit	Unsized Area	Clk spec Period	Optimized area	CPU (s)
C1355	2736	1.36	2744	1.4
		1.22	2774	2.2
		1.08	-	4.9
dalu	3414	2.1	3417	1.7
		1.88	3423	1.9
		1.22	3552	6.9
		1	-	13.3
C1908	2922	2.2	2929	1.6
		1.8	3091	8.6
		1.4	3554	23.3
		1	-	24.7
des	17368	2.8	17369	10.7
		2.4	17370	10.8
		2	-	17.3
C7552	10264	2.12	10321	11.8
		1.66	10614	44.3
		1.2	-	72.1

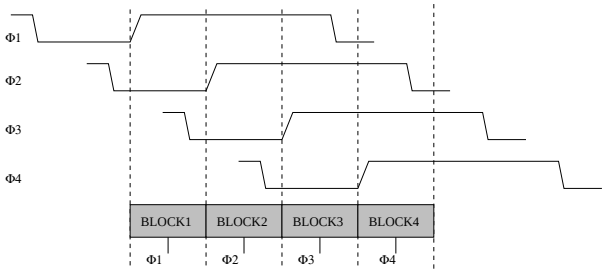


Fig. 10. Four-phase overlapping domino clocks

The objective of this tool is to provide a timing verification and long path correction tool for mixed static-domino circuits with arbitrary clocking strategy. For short path violations, delay insertion techniques [27] are more appropriate, and are not addressed in this work.

Although it is not possible to make a sweeping generalization, it is often true that domino logic circuits are easier to size than static logic: in a domino gate, usually the evaluation speed is critical, while both the rising and falling delays are important for static logic; additionally, the presence of inverters can provide the ability to drive large loads faster.

V. Conclusion

Domino logic is an attractive design style for high speed design. As its use becomes more widespread, there is a growing need for good EDA synthesis and optimization tools to support domino-based design. However, domino circuits are difficult to design since they are non-inverting, must obey strict timing constraints and are susceptible to noise. In this paper we have addressed several problems along a domino synthesis and optimization flow.

In most circuits, it is appropriate to use domino gates to speed up parts of the circuit, while the remainder is implemented in static CMOS. Therefore, we have addressed the problem of partitioning a circuit to determine which parts of the circuit should be implemented as static logic, and which parts as domino logic, in conformance with a specified clock scheme. We have presented partitioning algorithms for one-phase static domino partitioning and two-way domino partitioning, and extended these to two flows for two-phase static domino partitioning; all of these algorithms work to minimize a cost function under timing constraints. The results show that the effect of partitioning is to reduce the logic unating penalty, and find solutions with areas similar to the (typically lower) area of a pure static implementation and delays similar to the (typically faster) speeds of a pure domino implementation. The paper necessarily tackles only a subset of the problems encountered in static-domino partitioning, and several problems remain to be addressed. For example, if overlapping clocks are used, there is an additional tradeoff involved: the use of time borrowing can speed up the circuit, but at the cost of disallowing the insertion of inverting latches, thereby increasing the amount of logic duplication (however, logic duplication may be avoided for early arriving signals). We believe that the algorithms presented here provide a basis for a general framework for static-domino partitioning that can be modified to provide the solution to the partitioning problems if the clocking strategy is altered, or if the cost function is changed.

We have developed a timing verification and sizing optimization tool for circuits containing mixed domino and static logic. A timing analysis methodology is developed, and an optimization procedure based on TILOS is used to meet timing constraints and preserve adequate noise margins by constraining the sizing procedure.

Acknowledgements

The authors would like to gratefully acknowledge the numerous helpful suggestions made by the anonymous reviewers.

References

[1] G. Yee and C. Sechen, "Dynamic logic synthesis," in *Proceedings of the IEEE Custom Integrated Circuits Conference*, pp. 345–348, 1997.

[2] 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.

[3] M. R. Prasad, D. Kirkpatrick, and R. K. Brayton, "Domino logic synthesis and technology mapping," in *Workshop Notes, International Workshop on Logic Synthesis*, 1997.

[4] T. Thorp, G. Yee, and C. Sechen, "Domino logic synthesis using complex static gates," in *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design*, pp. 242–247, 1998.

[5] M. Zhao and S. S. Sapatnekar, "Technology mapping for domino logic," in *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design*, pp. 248–251, 1998.

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

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

[8] R. Puri, "Design issues in mixed static-domino circuit implementations," in *Proceedings of the IEEE International Conference on Computer Design*, pp. 270–275, 1998.

[9] K. Venkat, L. Chen, I. Lin, P. Mistry, and P. Madhani, "Timing verification of dynamic circuits," *IEEE Journal of Solid-State Circuits*, vol. 31, pp. 452–455, Mar. 1996.

[10] D. V. Campenhout, T. Mudge, and K. A. Sakallah, "Timing verification of sequential domino circuits," in *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design*, pp. 127–132, 1996.

[11] H. Y. Chen and S. M. Kang, "A new circuit optimization technique for high performance CMOS circuits," *IEEE Transactions on Computer-Aided Design*, vol. 10, pp. 670–676, May 1991.

[12] L. T. Wurtz, "An efficient scaling procedure for domino CMOS logic," *IEEE Journal of Solid-State Circuits*, vol. 28, pp. 979–982, Sept. 1993.

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

[14] W. H. Lien and W. P. Burleson, "Wave-domino logic: Theory and applications," *IEEE Transactions on Circuits and Systems*, vol. 42, pp. 78–90, Feb. 1995.

[15] J. Kernhof, M. Selzer, M. A. Beunder, B. Hoefflinger, B. Laquai, and I. Schindler, "Mixed static and domino logic on CMOS gate forest," *IEEE Journal of Solid-State Circuits*, vol. 25, pp. 396–402, Apr. 1990.

[16] R. R. Ortiz and M. C. Lefebvre, "Technology mapping for NORA dynamic logic circuits," in *Proceedings of the European Design Automation Conference*, pp. 310–314, 1993.

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

[18] J. L. Burns and J. A. Feldman, "C5M - A control-logic layout synthesis system for high-performance microprocessors," *IEEE Transactions on Computer-Aided Design*, vol. 17, pp. 14–23, Jan. 1998.

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

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

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

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

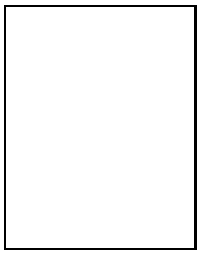
[23] H. Liu and D. F. Wong, "Network flow based circuit partitioning for time-multiplexed FPGA's," in *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design*, pp. 497–504, 1998.

[24] S. Iman, M. Pedram, C. Fabian, and J. Cong, "Finding unidirectional cuts based on physical partitioning and logic restructing," in *4th International Workshop on Physical Design*, 1993.

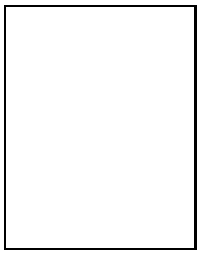
[25] J. P. Fishburn and A. E. Dunlop, "TILOS: A posynomial programming approach to transistor sizing," in *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design*, pp. 326–328, 1985.

[26] A. Dharchoudhury, D. Blaauw, J. Norton, S. Pullela, and J. Dunning, "Transistor-level sizing and timing verification of domino circuits in the PowerPC microprocessor," in *Proceedings of the IEEE International Conference on Computer Design*, pp. 143–148, 1997.

[27] N. V. Shenoy, R. K. Brayton, and A. L. Sangiovanni-Vincentelli, "Minimum padding to satisfy short path constraints," in *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design*, pp. 156–161, 1993.



Min Zhao received her B.S. in computer and its application, M.S. in electrical transmission and its automation, from Dalian Maritime University, Dalian, China, in 1993, 1996 and the Ph.D. degree in electrical engineering from University of Minnesota in 1999. She is currently an Electrical Engineer in Motorola, Austin, Texas. Her research interest is VLSI CAD for domino logic, technology mapping, power grid analysis.



Sachin Sapatnekar (S'86-M'93) received the B.Tech. degree from the Indian Institute of Technology, Bombay in 1987, the M.S. degree from Syracuse University in 1989, and the Ph.D. degree from the University of Illinois at Urbana-Champaign in 1992. From 1992 to 1997, he was an assistant professor in the Department of Electrical and Computer Engineering at Iowa State University. He is currently an associate professor in the Department of Electrical Engineering at the University of Minnesota.

He has coauthored two books, "*Design Automation for Timing-Driven Layout Synthesis*" (Kluwer Academic Publishers) and "*SequentialS Timing Analysis and Optimization*" (Kluwer Academic Publishers). He has served as an Associate Editor for the *IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing*. He is a recipient of the NSF Career Award and Best Paper awards at DAC97 and ICCD98.