Technology Mapping for Domino Logic *

Min Zhao Sachin S. Sapatnekar
Department of Electrical and Computer Engineering
University of Minnesota, 200 Union Street SE, Minneapolis MN 55455, USA.
{*zhaomin,sachin*}*@ece.umn.edu*

Abstract

Domino logic is a popular configuration for implementing high-speed circuits. An algorithm for domino logic mapping, under a parameterized library style, is presented here. Practical design methods, such as the use of multioutput domino and wide domino gates, are incorporated within the technology mapping framework. The technique can handle large circuits with small computational overheads, and shows improvements of up to about 37% over existing methods.

1 Introduction

The domino technology mapping problem is defined as: Given an optimized Boolean network and constraints on the width (maximum number of parallel chains) and height (maximum number of series transistors) of the domino gates, the nodes in the network are to be implemented with domino logic gates such that a cost objective is minimized. We examine this problem and explore an efficient parameterized library-based domino logic technology mapping method.

A parameterized library is defined as a collection of gates that satisfy the constraints on the width and height of the pull-down or pull-up implementations of a gate. For domino gates, since the number of possible cells in such a library is extremely large, the layout of the cells are produced by on-the-fly cell generation, instead of using a fixed cell library.

Technology mapping based on static standard library is a well established problem addressed in [1–3]. The cell-generator based mapping technique was originally proposed for static complex gate mapping in [4] and modified for domino logic in [5]. We use a dynamic programming approach to find the optimal solution.

Traditional technology mapping methods, originating with [1], have decomposed the directed acyclic graph (DAG) representing a circuit into a forest of trees of single-fanout nodes; each tree is then mapped to an available cell. However, this tree-by-tree mapping procedure may generate small trees, but domino gates can provide smaller area and delay only through forming larger complex gates. Therefore, multioutput domino gates and wide domino gate configurations are used in our approach. Instead of using only multifanout nodes or only primary output nodes as the roots of trees as in [1], we use both primary output nodes and reconvergent nodes. Depending on the cost tradeoff, the multifanout node inside the tree is either mapped with a multioutput gate, or is duplicated, or is mapped as a root of the domino gate tree.

*This work was supported in part by the National Science Foundation under contracts MIP-9502556 and MIP-9796305 and a gift from Intel Corporation.

2 Parameterized library mapping

Given an arbitrarily optimized network, it is first unated [6] and mapped into a two input AND-OR DAG network; this is then decomposed into two-input AND-OR trees. This is the starting point of our parameterized library mapping algorithm, which is based on dynamic programming [7].

2.1 Node structures and functions

Assume that the maximum constraint on the width and height of the domino gate are, respectively, W and H. At each node, we store the optimal subsolutions index for all possible [height,width] configuration from [1,1] to [H,W], and each such subsolution is referred to as a configuration. Therefore, there is a maximum of $H \times W$ optimal solutions that can be possibly stored for every node. Each optimal solution can be represented as $\{S, P, C, \{S_l, P_l\}, \{S_r, P_r\}\}$. Here, S, $(1 \leq S \leq H)$, is the height constraint of the current node, P, $(1 \leq P \leq H)$, is the width constraint of the current node and C is the area cost, measured in terms of the number of transistors; the objective in this paper is to minimize C. Different combinations of the child node configurations can lead to the same parent node configuration. We denote $\{S_l, P_l\}$, $\{S_r, P_r\}$ as the combination that provides the minimal cost for a configuration $\{S, P\}$.

Physically, $\{S, P\}$ represents a segment of a domino NMOS net, whose maximum pull-down width is P and

whose maximum pull-down height is S. The area cost is the accumulated area of its child transition cones, including the domino gate input cost and the number of transistors in the current domino circuit segment.

Due to the series-parallel structure of the domino pulldown, only two types of nodes in the pulldown structure need to be considered: AND nodes and OR nodes. An AND node corresponds to a series connection, while an OR node corresponds to a parallel connection in the series-parallel subgraph. The formulas for updating the value of S and P while combining two child configurations are similar to [4]:
1. OR node: $S = \max(S_l, S_r)$, $P = P_l + P_r$
2. AND node: $S = S_l + S_r$, $P = \max(P_l, P_r)$
3. LEAF node or GATEFORM node: $S = 1$, $P = 1$
A leaf node corresponds to a primary input, and a gate formation node, GATEFORM, corresponds to a situation where the partial solution so far (during the dynamic programming procedure) is condensed into a domino gate with an output at that node, and a new domino structure is started from that point on.

The area cost function, C, is measured as the number of transistors for a configuration. For each of the above possibilities, as a function of the costs C_l and C_r, respectively, of the left and right child subtrees, is as shown below.
1. OR or AND node: The number of transistors is found by simply summing up the number in the subtrees, i.e., $C = C_l + C_r$.
2. LEAF node: At a leaf node, $C = 1$.
3. GATEFORM node: $C = C_{minimal} + 5$.
The first two cases correspond to the partial pulldown structure (not counting the transistors connected to the clock) constructed so far for the current configuration. The last case corresponds to the addition of two transistors connected to the clock nodes and two transistors in the output node, besides the transistor in the newly created gate that is connected to the output of this newly-created gate.

2.2 Node mapping algorithm

```
for each valid [H,W] of the left child {
    for each valid [H,W] of the right child {
        {S,P} = COMBINE({S_l,P_l},{S_r,P_r}) ;
        if {S,P} satisfies the constraints (H,W) {
           C = COST FUNCTION(C_l,C_r)
           if (C < C[S,P]_minimal) C[S,P]_minimal = C .
           if (C < C_minimal) C_minimal = C .
        }
    }
}
C[1,1] = GATEFORM(C_minimal)
```

Here, C_l is the left child's optimal solution under the constraint $\{S_l, P_l\}$ while C_r is the right child's optimal solution under the constraint $\{S_r, P_r\}$. The solution $C[1,1]$ of the root of the decomposed tree is the optimal solution of the subject AND-OR tree.

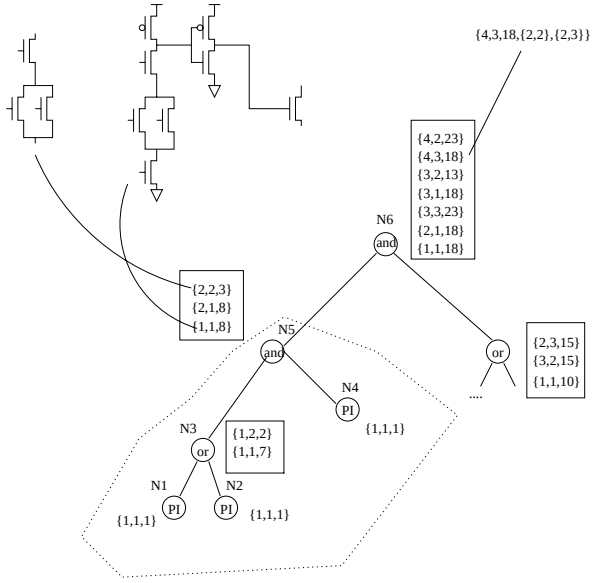

Figure 1: Parameterized library mapping.

This procedure is illustrated in Figure 1. In this example, three-tuples are used to represent configurations $\{S, P, C\}$. All primary inputs is initialized with tuple $\{1, 1, 1\}$. The tree is traversed from the leaf node upwards, and all solutions are enumerated using dynamic programming, eliminating any solutions that are suboptimal. From the dynamic programming viewpoint, the optimal solution under the constraint $\{S, P\}$ is an optimal substructure. The nonsuboptimal solutions at a node are listed and the problem is solved by recursively enumerating these for higher level nodes of the tree. For example, the operation of COST FUNCTION on both $(\{2, 2, 3\}, \{2, 3, 15\})$ and on $(\{2, 1, 8\}, \{2, 3, 15\})$ produces a $\{4, 3, C\}$ configuration. Only the 3-tuple of this type with minimal cost, $\{4, 3, 18\}$, and its corresponding child tuples are stored at node N6 as a partial solution that may be used in future. For the configuration of $S = P = 1$, we choose the minimum cost solution at that node and construct a gate corresponding to that solution. At node N6, the minimal cost obtained from all combinations of its children is 13, and the solution $\{1, 1, 18\}$ is obtained using the formula $C = C_{minimal} + 5$.

From the above procedure, we can see that the space complexity of the algorithm is $O(WHN)$ and the time complexity is $O(W^2H^2N)$, where N is number of nodes. At each node, the AND-OR cost function will be executed at most $W^2H^2/2$ times, but gen-

erally the number of executions is much lower than this value. From the statistics of 16 ISCAS85 benchmarks, the average cost addition operation at every node is approximately 4 for the tree-by-tree method.

3 Multioutput gate mapping

One of the domino gate configurations used by our mapper is the wide dynamic AND gate [8], which can implement a stack of six serial NMOS transistors with two stacks of three serial transistors. A second design style is the multioutput domino gate, an example of which is shown in Figure 2; these are commonly used in high speed adder design [9,10]. In our mapper, we use separate pull-up transistors for each multioutput branch, as shown in Figure 2, to prevent problems with sneak paths.

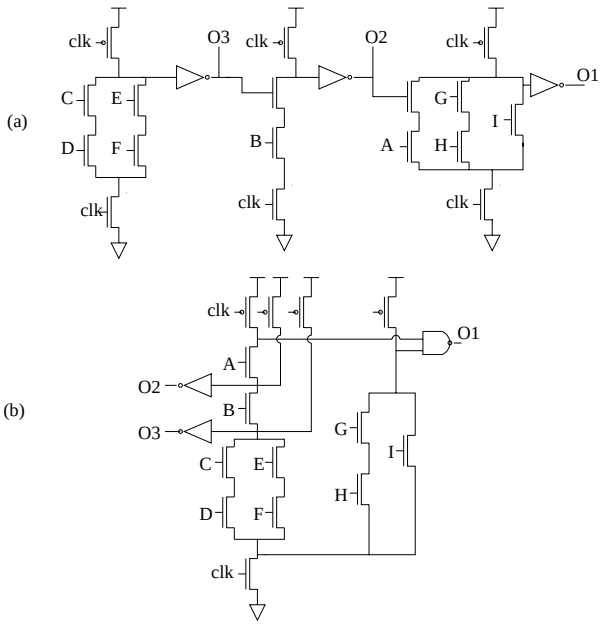

Figure 2: Implementation of a multioutput gate.

Traditionally, tree decomposition is carried out by decomposing trees by introducing breaks at multifanout points. However, this can create very small trees. This is an especial liability for domino gates since they have the overhead of the clock transistors and the output inverter and are more effective on larger subcircuits. Fortunately, the domino logic style can support multioutput gates and can therefore naturally handle multifanout points in the DAG. Therefore, in our mapper, we decompose the DAG into trees whose roots are primary outputs and reconvergent nodes.

In domino logic, the duplication cost of a single input is one NMOS transistor. In addition, multioutput logic can be implemented with a single multioutput domino gate by sharing common substructures. The additional cost associated with this kind of gate is a pull up transistor and the transistors in the output inverting logic. Each such possibility is a potential mapping scheme for the multifanout node during postorder mapping. Hence, a multifanout node will possibly be duplicated, or mapped as a multioutput gate, or be mapped as a simple domino gate at the node, depending on the cost tradeoff. The use of duplication and of multioutput domino also has the advantage that there are fewer gate levels than would be obtained by forming a gate at each multifanout gate.

4 Implementation and results

The parameterized library mapping algorithm has been implemented in C++. The input circuits for the technology mapping are unate equivalents of the benchmark circuits that were obtained by first optimizing with SIS, then pushing the inverters as close to the inputs as possible, and finally duplicating the fanin cones of the inverters.

Table 1: Comparison with the results of [5].

Circuits	Our approach area/level	Approach of [5] area/level	Reduction %
c8	265/4	328/7	23.8 %
count	283/6	348/16	23 %
i6	830/2	890/3	7.2 %
C880	1096/10	1499/7	36.8 %
C499	1752/8	1846/12	5.4 %
dalu	2031/12	2142/15	5.5 %

The results, compared with paper [5], are shown in Table 1. Both techniques use W=6, H=4 (W=H=4 for dalu) as the constraints of the parameterized library. In our work, we have not explicitly tried to minimize the replicated logic while pushing the inverters back during unate optimization. We expect that we will have further reductions in the transistor count if we implement such an optimization, e.g. [5,6].

Table 2 serves several objectives and shows the comparisons of our work with several other possible approaches that aim at area minimization. We use W = H = 4 as the width and height constraints. The results are presented and compared in terms of both area and delay, where the delay is estimated by a coarse measure that counts the number of gate levels.

Columns 2 and 3 contains the CPU time and results obtained from the tree-by-tree mapping approach combined with the node mapping method of section 2. Column 4 shows the results obtained from the mapping algorithm of section 3 combined with the node mapping method. The results in column 4 are com-

pared with those obtained from SIS, using a script file similar to *script.domino* [5] to optimize the benchmark circuits applying the library `44-3.genlib`. The CPU times are not shown, but are all under 7 seconds. A comparison with the results of static mapping with SIS is shown in columns 6 and 7, showing that depending on the circuit, either static mapping or domino mapping may provide a smaller area, but domino logic always has an advantage in terms of the number of levels. The area disadvantage, when it is present, arises primarily because of the area overhead required to move the inverters back towards the primary inputs, which may necessitate logic replication.

This motivated us to study the efficiency of our domino technology mapping method, neglecting the efficiency of unating. As an experiment, we fed the unate input circuit used for the domino technology mapping to the SIS static mapping, by using `map -m 0`. The SIS mapping results are shown in columns 8–10 and it is seen that our method presents a considerable advantage.

5 Conclusion

In this paper, we have explored the new mapping technique for domino logic, including optimal parameterized library mapping and multifanout domino gates mapping. The area minimization mapping can be extended to minimize delay and power, as well as cost minimization mapping under timing constraints.

References

[1] K. Keutzer, "DAGON: technology mapping and local optimization," in *Proc. ACM/IEEE Design Automation Conf.*, pp. 341–347, 1987.

[2] E. Detjens, G. Gannot, R. Rudell, A. Sangiovanni-Vincentelli, and A. Wang, "Technology mapping in MIS," in *Proc. IEEE/ACM Int. Conf. Computer-Aided Design*, pp. 116–119, 1987.

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

[4] M. R. C. M. Berkelaar and J. A. G. Jess, "Technology mapping for standard-cell generators," in *Proc. IEEE/ACM Int. Conf. Computer-Aided Design*, pp. 470–473, 1988.

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

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

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

[8] T. Williams, "Dynamic Logic: Clocked and Asynchronous," Tutorial notes at the Int. Solid State Circuits Conf., 1996.

[9] J. Wang, Z. D. Wang, G. A. Jullien, and W. C. Miller, "Area-time analysis of carry lookahead adders using enhanced multiple output domino logic," in *Proc. IEEE Int. Symp. Circuits Syst.*, pp. 59–62, 1994.

[10] Z. Wang, G. A. Jullien, W. C. Miller, J. Wang, and S. S. Bizzan, "Fast adders using enhanced multiple-output domino logic," *IEEE J. Solid-State Circuits*, vol. 32, pp. 206–213, Feb. 1997.

Table 2: Comparison of simple mapping, multioutput mapping and SIS results

Column 1	Column 2	Column 3	Column 4	Column 5	Column 6	Column 7	Column 8	Column 9	Column 10
Circuits	CPU time(s)	simple mapping	multi-output mapping	reduction %	SIS orig input	reduction %	SIS unate input	reduction %	SIS time (s)
C1355	0.6	2575/9	1776/8	45 %	1418/19	-20.2 %	2264/16	27.5 %	71
dalu	0.7	2045/11	2031/12	0.7 %	2602/26	28.1 %	2836/16	39.6 %	118
C880	0.3	1198/13	1136/12	5.5 %	1038/19	-8.6 %	1486/22	30.1 %	57
i6	0.3	830/2	830/2	0 %	1370/3	65 %	1378/4	66.0 %	53
count	0.1	353/16	317/12	11.4 %	354/18	11.7 %	454/30	43.2 %	14
c8	0.1	270/4	268/4	0.8 %	338/8	26.1 %	392/6	46.3 %	16
C1908	0.5	2148/15	1923/12	11.7 %	1390/25	-27.7 %	2354/22	22.4 %	78
C2670	0.7	1926/9	1859/10	3.6 %	1908/18	2.6%	2680/14	44.2 %	108
C3540	1.4	4583/18	4378/17	4.7 %	3056/30	-30.2 %	5826/30	33.1 %	251
C6288	4.0	12313/70	12277/58	0.3 %	7532/120	-38.7%	15986/100	30.2 %	577
b9	0.1	229/4	207/3	10.6 %	318/7	53.6 %	332/6	60.4 %	14
k2	1.7	5989/11	4859/8	23.3 %	4462/12	-8.17 %	5988/14	23.2 %	288
des	3.6	10005/11	9885/11	1.2 %	11194/18	13.2 %	14306/18	44.7 %	639
C7552	2.6	8153/ 16	7221/15	12.9 %	5932/29	-17.8 %	9030/28	25.1 %	302
t481	1.4	3758/11	3636/10	3.4 %	4004/15	10.1 %	4096/16	12.7 %	351
rot	0.6	1706/10	1686/9	1.2 %	1734/21	2.9%	2384/16	41.4 %	97