An Improved Graph-Based FPGA Technology Mapping Algorithm For Delay Optimization

Jason Cong, Yuzheng Ding, Andrew B. Kahng, Peter Trajmar[1]
Department of Computer Science
University of California, Los Angeles, CA 90024
Kuang-Chien Chen
Fujitsu America, Inc.
3055 Orchard Drive, San Jose, CA 95134

Abstract

We present a graph based technology mapping algorithm, called DAG-Map, for delay optimization in lookup-table based FPGA designs. Our algorithm carries out technology mapping and delay optimization on the entire Boolean network, instead of decomposing it into fanout-free trees and mapping each tree separately as in most previous algorithms. As a preprocessing step, we introduce a general algorithm which transforms an arbitrary n-input network into a two-input network with only $O(1)$ factor increase in the network depth; previous transformation procedures may result in an $\Omega(\log n)$ factor of increase. We also present a graph matching based technique used as a postprocessing step which optimizes the area without increasing the delay. We tested the DAG-Map algorithm on the MCNC logic synthesis benchmarks. Compared with previous algorithms (Chortle-d and MIS-pga), DAG-Map reduces both the network depth and the number of lookup-tables.

1. Introduction

The technology mapping problem is to implement a synthesized Boolean network using logic cells from a prescribed cell family. Much work has been done on the technology mapping problem for conventional gate array or standard cell designs (e.g. [12,4].) However, these methods do not apply immediately to the technology mapping problem for lookup-table based FPGAs [19] since a K-input lookup-table (K-LUT) can implement any one of 2^{2^K} K-input logic gates, and consequently the equivalent cell family is too large to be manipulated efficiently.

Recently, a number of technology mapping algorithms have been proposed for area optimization in lookup-table based FPGA designs. These include the MIS-pga program developed by Murgai *et al.* [14] and its improved version MIS-pga (new) [16], the Chortle program and its successor Chortle-crf, developed by Francis *et al.* [5,7], the Xmap program developed by Karplus [11], and an algorithm proposed by Woo [18]. The objective of these algorithms is to minimize the number of programmable logic blocks in the mapping solution.

Previous work on FPGA mapping for delay optimization consists of Chortle-d, developed by Francis *et al.* [6], and the MIS-pga (delay) extension, developed by Murgai *et al.* [15]. The Chortle-d algorithm first decomposes the network into fanout-free trees and then uses dynamic programming and bin-packing heuristics to map each tree independently. The objective at each step is to minimize the depth of the node being processed. The MIS-pga extension for delay optimization contains two phases, mapping and placement/routing. Various decomposition techniques are used to dynamically resynthesize the network to reduce the delay and area. The advantage of this approach is that it takes layout information into consideration at the technology mapping stage.

In this paper, we present a graph based technology mapping algorithm, called DAG-Map, for delay optimization in lookup-table based FPGA design. Our algorithm carries out technology mapping and delay optimization on the entire Boolean network without decomposing it into trees. Our algorithm is optimal for trees for any K-LUTs while Chortle-d is optimal for trees only when $K \leq 6$ [6]. Moreover, as a preprocessing step, we introduce a general algorithm for transforming an arbitrary n-node network into a two-input network with only an $O(1)$ factor of increase in the network depth, while the previous transformation procedure may result in an $\Omega(\log n)$ factor of increase. We also present a matching based technique used as a postprocessing step for area optimization without increasing network delay. We tested DAG-Map on a set of MCNC logic synthesis benchmarks and compared it with Chortle-d and MIS-pga (delay). Experimental results showed that on average, DAG-map reduces both the network delay and the number of lookup-tables.

2. Problem Formulation

A Boolean network can be represented as a directed acyclic graph (DAG) where each node represents a logic gate and there is a directed edge (i, j) if the output of gate i is an input of gate j. A primary input (PI) node has no incoming edge and a primary output (PO) node has no outgoing edge. We use $input(v)$ to denote the set of nodes which supply inputs to gate v. Given a subgraph H of the Boolean network, $input(H)$ denotes the set of *distinct* nodes which supply inputs to the gates in H. For a node v in the network, a *K-feasible cone at* v, denoted C_v, is a subgraph consisting of v and predecessors of v such that any path connecting a node in C_v and v lies entirely in C_v and $|input(C_v)| \leq K$. The *level* of a node v is the length of the longest path from any PI node to v. The *depth* of a network is the largest node level in the network.

We assume that each programmable logic block in an FPGA is a K-LUT that can implement any K-input Boolean function. Thus, each K-LUT can implement any K-feasible cone of a Boolean network. The technology mapping problem for K-LUT based FPGAs is to cover a given Boolean network with K-feasible cones[2]. A

[1] Current address: Zycad Corporation, 47100 Bayside Parkway, Fremont, CA 94538.

[2] We do not require the covering to be disjoint since we allow nodes in the network to be replicated, if necessary, as long as the resulting network is logically equivalent to the original one.

1063-6404/92 $3.00 © 1992 IEEE

technology mapping solution S is a DAG where each node is a K-feasible cone (equivalently, a K-LUT) and the edge (C_u, C_v) exists if u is in $input(C_v)$. We want to compute a mapping solution that results in small circuit delay and, secondarily, small chip area. The delay of a FPGA circuit is determined by the delay in K-LUTs and delay in the interconnection paths. Since layout information is not available at this stage, we assume that each edge in the mapping solution S contributes a constant delay. Hence, the circuit delay is determined by the depth of S, since each K-LUT contributes a constant delay (the SRAM access time) independent of the function it implements. Therefore, the main objective of our algorithm is to determine a mapping solution S with depth as small as possible. Our secondary objective is to minimize the number of K-LUTs in the mapping solution.

3. The DAG-Map Algorithm

Our DAG-Map algorithm consists of three major steps. The first step transforms an arbitrary Boolean network into a two-input network. The second step maps the two-input network into a K-LUT FPGA network for delay minimization. The third step performs area optimization of the FPGA network without increasing the network delay.

3.1. Transforming Arbitrary Networks into Two-Input Networks

As in [7,6], we assume that each node in the given Boolean network is a simple gate (i.e. AND, OR, NAND, or NOR gate).[3] The first step of our algorithm is to transform the given Boolean network of simple gates into a two-input network (i.e. each gate in the network has at most two inputs). There are two reasons for carrying out such a transformation. First, we want to limit the number of inputs of each gate to be no more than K so that we do not have to decompose gates during technology mapping. Second, if we think of FPGA technology mapping as a process of packing gates in a given network into K-LUTs, then, intuitively, smaller gates will be more easily packed, with less wasted space in each K-LUT.

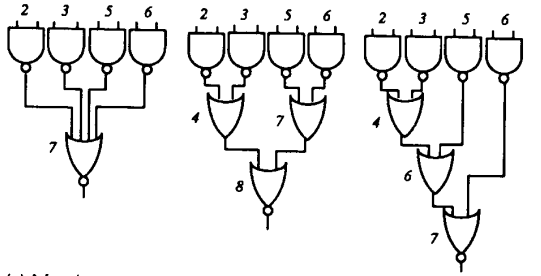


Fig. 1 Transforming multi-input network into two-input network (numbers indicate levels)

[3] A complex gate can be represented in sum-of-products form and then replaced by two levels of simple gates. We use the technology decomposition command `tech_decomp -o 1000 -a 1000` in MIS [1] to realize such a transformation.

A straightforward way to transform an n-node arbitrary network into a two-input network is to replace each m-input gate ($m \geq 3$) by a balanced binary tree. Fig. 1(a) shows a 4-input gate v (where the numbers beside the nodes indicate their levels) and Fig. 1(b) shows the result of replacing it by a balanced binary tree. We see that the level of v increases from 7 to 8. In general, such a straightforward transformation may increase the network depth by an $\Omega(\log n)$ factor [2]. However, if we replace v by the binary tree shown in Fig. 1(c), the level of v remains 7. Therefore, the objective of this step is to choose a binary tree to replace each multiple-input node so that the height of the resulting network is as small as possible.

Given an arbitrary Boolean network G, we transform G into a two-input network G' as follows. We process all the nodes in G in topological order starting from PI nodes. For each multiple-input node v, we construct a binary tree $T(v)$ to replace v using an algorithm similar to Huffman's algorithm for constructing a prefix code of minimum average length [10]. Writing $input(v) = \{u_1, u_2, ..., u_m\}$, we note that nodes $u_1, u_2, ..., u_m$ have been processed before v and their levels in the new network G' are fixed. Intuitively, we want to combine nodes with smaller levels first when we construct the binary tree $T(V)$. Our algorithm is as follows.

Algorithm: decompose-multi-input-gate (DMIG)
let $V = \{u_1, u_2, ..., u_m\}$;
while $|V| \geq 2$ **do**
  select two nodes u_i and u_j in V with smallest levels;
  introduce a new node x;
  $input(x) = \{u_i, u_j\}$;
  $level(x) = \max(level(u_i), level(u_j)) + 1$;
  $V = (V - \{u_i, u_j\}) \cup \{x\}$;
end-while
the root of $T(v)$ = the only node left in V;
end-algorithm.

If we apply the DMIG algorithm to the example in Fig. 1(a), we obtain the binary tree shown in Fig. 1(c). In general, we have the following results:

Theorem 1 For an arbitrary Boolean network G of simple gates, let G' be the network obtained by applying the DMIG algorithm to each multi-input gate in a topological order starting from the PI nodes. Then, we have

$$depth(G') \leq \log(2d-1) \cdot (depth(G) - 1) + \log 2I$$

where d is the maximum degree of fanout in G and I is the number of PI nodes in G. □

The proof is given in [2]. A similar result was obtained by Hoover *et al.* for bounding fanout in a Boolean network [9]. Since in practice d is bounded by a constant (the fanout limit of any output), the depth of the two-input network G' is increased by just a small *constant* factor $\log(2d-1)$ away from $depth(G)$.[4] A pathological example for the balanced binary tree based transformation can be found in [2], where the balanced binary tree based transformation increases the network depth by an $\Omega(\log n)$ factor even when $d=1$.

[4] Here we assume that $depth(G) = \Omega(\log 2I)$, which is true for most networks in practice. This excludes the unrealistic case where $\log 2I$ is the dominating term in the right-hand side of the inequality in Theorem 1.

3.2. Technology Mapping for Delay Minimization

After we obtain a two-input Boolean network, we carry out technology mapping directly on the entire network. We use a method similar to that of Lawler *et al.* for module clustering to minimize delay in Boolean networks [13]. The mapping is performed in two steps: we first label the network to determine the level of each node in the final mapping solution, and then we generate the logically equivalent network of K-LUTs.

The first step assigns a label $h(v)$ to each node v of the two-input network, with $h(v)$ equal to the level of the K-LUT containing v in the final mapping solution. Clearly, we want $h(v)$ to be as small as possible in order to achieve delay minimization. We label the nodes in a topological order starting from the PI nodes. The label of each PI node is zero. If node v is not a PI node, let p be the maximum label of the nodes in $input(v)$. We use $N_p(v)$ to denote the set of predecessors of v with label p. Then, if $input(N_p(v) \cup \{v\}) \le K$, we assign $h(v) = p$; otherwise, we assign $h(v) = p + 1$. With this labeling, it is evident that $N_{h(v)}(v)$ forms a K-feasible cone at v for each node v in the network[5].

The second step uses the labeling to generate K-LUTs in the mapping solution. Let L represent the set of outputs which are to be implemented using K-LUTs. Initially, L contains all the PO nodes. We process the nodes in L one by one. For each node v in L, we remove v from L and generate a K-LUT v' to implement the function of gate v such that $input(v') = input(N_{h(v)}(v))$. (Recall that $N_{h(v)}(v)$ forms a K-feasible cone at v.) Then, we update the set L to be $L \cup input(v')$. The second phase stops when L consists of only PI nodes in the original network. Clearly, we obtain a network of K-LUTs logically equivalent to the original network. The algorithm can be summarized as follows.

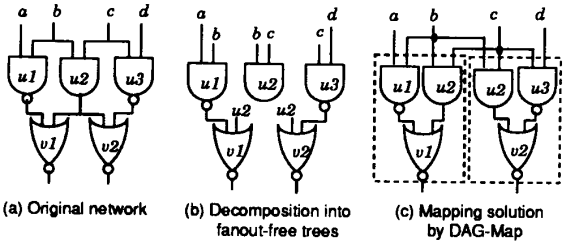
```
Algorithm: DAG-Map
 /* phase 1: labeling the network */
 for each PI node v do
   h(v) = 0;
 T = list of non-PI nodes in topological order;
 while T is not empty do
   remove the first node v from T;
   let p = max{h(u) | u ∈ input(v)};
   if |input(N_p(v) ∪ {v})| ≤ K then h(v) = p
   else h(v) = p + 1;
 end-while
 /* phase 2 : generate K-LUTs */
 L = list of PO nodes;
 while L contains non-PI nodes do
   remove a non-PI node v from L, i.e. L = L - {v};
   introduce a K-LUT v' to implement the function of v
     such that input(v') = input(N_h(v)(v));
   L = L ∪ input(v');
 end-while
end-algorithm.
```

The DAG-Map algorithm has several advantages:

(1) It works on the entire network without decomposing it into fanout-free trees; this usually leads to better mapping solutions. For example, decomposing the two-input network shown in Fig. 2(a) into fanout-free trees (as shown in Fig. 2(b)) yields a two-level mapping solution with three 3-LUTs. However, the DAG-Map algorithm gives an one-level mapping solution with two 3-LUTs (as shown in Fig. 2(c)).

(2) The DAG-Map algorithm can replicate nodes, if necessary, to minimize the network delay in the mapping solution. For the solution shown in Fig. 2(c), node u_2 is replicated to get an one-level mapping solution. Note that if node u_2 is not replicated, the depth of the mapping solution is at least two.

(3) The DAG-Map algorithm is optimal when the initial network is a tree (or a set of trees).



(a) Original network (b) Decomposition into fanout-free trees (c) Mapping solution by DAG-Map

Fig. 2 A mapping example for $K=3$

Theorem 2 For any integer K, if the given Boolean network is a tree with fanin no more than K at each node, the DAG-Map algorithm produces a minimum depth mapping solution for K-LUT based FPGAs. □

The proof is included in[2]. Recall that a similar result was shown for Chortle-d in [6], but it holds only for $K \le 6$ since the bin-packing heuristics are no longer optimal for $K > 6$.[6]

Although the DAG-Map algorithm is optimal for trees, it may not be optimal for general networks. In general, DAG-Map is optimal if the mapping constraint for each programmable logic block is monotone. A constraint X is *monotone* if a network H satisfying constraint X implies that any subgraph of H also satisfies X [13]. For example, if we limit the number of gates a programmable logic block may cover, it will be a monotone constraint. Unfortunately, the constraint on the number of distinct inputs of each programmable logic block is monotone only for trees. Pathological examples can be found in [2].

3.3. Area Optimization Without Increasing Delay

After we obtain a mapping solution of minimum depth, we apply two operations for area optimization as a post-processing step. These operations reduce the number of K-LUTs in the mapping solution without increasing the network depth. In this sub-section, each node is a K-LUT in the mapping solution.

The first operation is *gate decomposition*, inspired by the gate decomposition concept used in Chortle-crf [7]. The basic idea is as follows. If a node v implements a simple gate of multiple inputs in the mapping solution,

[5] Note that $v \in N_{h(v)}(v)$ since v is a predecessor of itself.

[6] However, Chortle-d does not require that each node in the tree has degree no more than K.

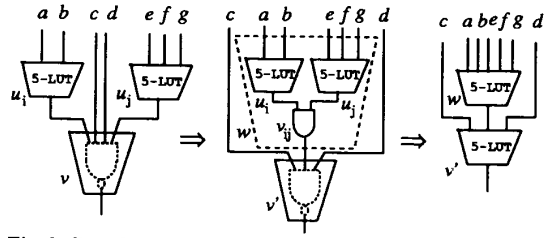


Fig. 3 Gate decomposition for area optimization *(K=5)*

then for any two of its inputs u_i and u_j, if $|input(u_i) \cup input(u_j)| \leq K$, we can decompose v into two nodes v_{ij} and v' of the same type as v, such that $input(v_{ij})=\{u_i, u_j\}$ and $input(v')=input(v) \cup \{v_{ij}\}-\{u_i, u_j\}$, and implement u_i, u_j and v_{ij} using one K-LUT, so that the number of K-LUTs is reduced by one. Such a decomposition is logically equivalent because of the associativity of the simple functions. Moreover, the decomposed node v has one fewer input (which is beneficial to subsequent area optimization operations). Figure 3 illustrates the gate decomposition operation.

This method can be generalized to the case where the decomposed node v implements a complex function. We use Roth-Karp decomposition [17] to determine if the node can be feasibly decomposed to v_{ij} and v' as in the preceding paragraph. Although Roth-Karp decomposition runs in exponential time in general, it takes only constant time in our algorithm, since the number of fanins of a K-LUT is bounded by the small constant K. If there is a pair of inputs u_i and u_j of node v on which Roth-Karp decomposition succeeds, and $|input(u_i) \cup input(u_j)| \leq K$, then the gate-decomposition operation is applicable, and we say that u_i and u_j are *mergeable*.

Another postprocessing operation for area optimization is called *predecessor packing*. For a node v, if an input node u_i of v satisfies $|input(v) \cup input(u_i)| \leq K$, then v and u_i are merged into a single K-LUT. In this case, we say v and u_i are *mergeable*. This operation reduces the number of K-LUTs by one. Fig. 4 shows an example of predecessor packing.

There are usually many pairs of mergeable nodes in a network, but not all of these merge operations can be performed at the same time. We thus use a graph matching approach to avoid merging nodes in arbitrary order; this achieves a globally good result. We construct an undirected graph $G=(V,E)$, where the vertex set V represents the nodes of the network, and an edge (v_i, v_j) is in the edge set E if and only if v_i and v_j are mergeable. Clearly a maximum cardinality matching in G corresponds to a maximum set of merging operations that

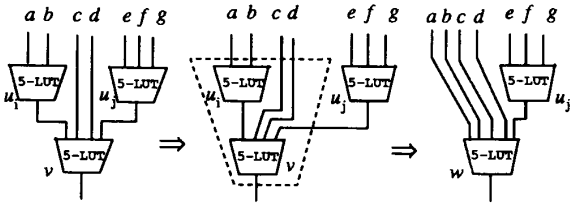


Fig. 4 Predecessor packing for area optimization *(K=5)*

can be applied simultaneously. Therefore we find a maximum matching in G and apply the merging operations corresponding to the matched edges simultaneously. We then reconstruct the graph G for the reduced network and repeat the above procedure until we are unable to construct a non-empty E. The experimental results show that this matching based merging algorithm usually converges after only one or two iterations. The maximum graph matching problem can be solved in $O(n^3)$ time [8].

4. Experimental Results

We have implemented the DAG-Map algorithm and tested it on a set of MCNC benchmark examples. We chose $K = 5$. For each benchmark example, we first used a standard MIS script (used by Chortle-d [6]) on the initial network to perform area minimization. Next, we applied the DMIG algorithm to transform the network into a two-input network. We then used DAG-map to map into a 5-LUT network. Finally, the matching based postprocessing step was performed.

Table 1 shows the comparison of the results of our algorithm with those of the Chortle-d algorithm (quoted from [6]) and those of the mapping phase of MIS-pga delay optimization algorithm (quoted from [15]). Overall, the solutions of Chortle-d used 59% more lookup-tables and had 2% larger network depth, and the solutions of MIS-pga (delay) used 3% more lookup-tables and had 6% larger network depth. In all cases, the running time of our algorithm is no more than 100 seconds on a Sun Sparc IPC (15.8MIPS).

In order to judge the effectiveness of our DMIG algorithm for transforming the initial network into a two-input network, We also compared the DMIG algorithm with the transformation procedure in MIS (specifically, the MIS command `tech_decomp -a 2 -o 2`). In Table 2, the first four columns compare the results of the two transformation algorithm, while the last four columns compare the results of DAG-Map on different two-input networks produced by the two algorithms. In all cases, the DMIG procedure resulted in smaller or the same depths in the 5-LUT networks, and on average it used

Technology Mapping for 5-LUT FPGAs									
	chortle-d			MIS-pga (d)			DAG-Map		
	LUTs	dpt	time	LUTs	dpt	time	LUTs	dpt	time
5xp1	26	3	0.1	21	2	3.5	22	3	1.1
9sym	63	5	0.2	7	3	15.2	60	5	2.3
9symml	59	5	0.1	7	3	9.9	58	5	2.3
C499	382	6	1.8	199	8	58.8	68	4	12.1
C880	329	8	0.9	259	9	39.0	130	8	6.1
alu2	227	9	0.7	122	6	42.6	156	9	7.6
alu4	500	10	0.3	155	11	15.4	274	10	17.0
apex6	308	4	0.8	274	5	60.0	246	5	11.1
apex7	108	4	0.2	95	4	8.4	83	4	2.8
count	91	4	0.1	81	4	5.1	31	5	0.8
des	2086	6	9.2	1397	11	937.8	1427	5	92.1
duke2	241	4	0.4	164	6	16.4	181	4	5.5
misex1	19	2	0.1	17	2	1.7	19	2	0.8
rd84	61	4	0.2	13	3	9.8	49	4	2.7
rot	326	6	1.0	322	7	50.0	251	7	13.7
vg2	55	4	0.1	39	4	1.7	29	3	0.9
z4ml	25	3	0.1	10	2	2.1	5	2	0.4
total	4906	87	16.3	3182	90	1277.4	3089	85	179.3
compr.	+59%	+2%	-	+3%	+6%	-	1	1	-

Table 1. Comparison with Chortle-d and MIS-pga (delay).

Comparison of 2-Input Network Transformation Methods								
	Before Mappings				After 5-LUT Mappings			
	mis tech_decomp		DMIG algo		mis tech_decomp		DMIG algo	
	gates	depth	gates	depth	gates	depth	gates	depth
5xp1	88	9	88	9	22	3	22	3
9sym	201	16	201	13	65	5	60	5
9symml	199	17	199	13	61	5	58	5
C499	392	25	392	25	66	4	68	4
C880	347	37	347	35	131	8	130	8
alu2	371	36	371	31	159	10	156	9
alu4	664	40	664	34	263	11	274	10
apex6	651	16	651	15	250	6	246	5
apex7	201	14	201	13	80	4	83	4
count	112	20	112	19	31	5	31	5
des	3049	19	3049	16	1461	6	1427	5
duke2	325	16	325	11	177	5	181	4
misex1	49	6	49	6	19	2	19	2
rd84	153	14	153	11	44	4	49	4
rot	539	27	539	21	256	7	251	7
vg2	72	15	72	10	29	4	29	3
z4ml	27	10	27	10	5	2	5	2
total	7440	337	7440	292	3119	91	3089	85
compr.	+0%	+15%	1	1	+1%	+7%	1	1

Table 2. Comparison of 2-input network transformations.

Effectiveness of Post-Processing Step for Depth Minimized 5-Input Lookup Table Mappings				
	original		after post-processing	
	LUTs	depth	LUTs	depth
5xp1	25	3	22	3
9sym	76	5	60	5
9symml	68	5	58	5
C499	80	4	68	4
C880	137	8	130	8
alu2	169	9	156	9
alu4	301	10	274	10
apex6	313	5	246	5
apex7	101	4	83	4
count	43	5	31	5
des	1674	5	1427	5
duke2	196	4	181	4
misex1	20	2	19	2
rd84	51	4	49	4
rot	275	7	251	7
vg2	32	3	29	3
z4ml	5	2	5	2
total	3566	85	3089	85
compr.	+15%	+0%	1	1

Table 3. Effect of the postprocessing for area minimization.

fewer 5-LUTs.

Finally, we tested the effectiveness of our postprocessing procedure for area optimization. The results are shown in Table 3. After the postprocessing step, the total number of lookup-tables is reduced by 15%.

5. Conclusions

We have presented a graph based technology mapping algorithm for delay optimization in lookup-table based FPGA design. Our algorithm consists of three main steps: transformation of an arbitrary network into a two-input network, technology mapping on the entire two-input network for delay minimization, and matching based global area optimization in the mapping solution. We have tested our algorithm on a large set of benchmark examples and achieved satisfactory results.[7]

6. Acknowledgments

We thank Dr. Masakatsu Sugimoto for his support of this research. This work was also supported by the NSF under grants MIP-9110511 and MIP-9110696. We thank Professor Jonathan Rose and Robert Francis for their assistance to our comparative study.

References

[1] Brayton, R. K., R. Rudell, and A. L. Sangiovanni-Vincentelli, "MIS: A Multiple-Level Logic Optimization," *IEEE Transactions on CAD*, pp. 1062-1081, November 1987.

[2] Chen, K. C., J. Cong, Y. Ding, A. B. Kahng, and P. Trajmar, "DAG-Map: Graph-based FPGA Technology Mapping for Delay Optimization," *IEEE Design and Test of Computers* , 1992, to appear.

[3] Cong, J. and Y. Ding, "An Optimal Technology Mapping Algorithm for Delay Optimization in Lookup-Table Based FPGA Designs," *Proc. IEEE Int'l Conf. on Computer-Aided Design*, 1992, to appear.

[7] After this work was submitted, Cong and Ding recently showed that the LUT-based FPGA technology mapping problem for delay optimization can be solved optimally in polynomial time [3].

[4] Detjens, E., G. Gannot, R. Rudell, A. Sangiovanni-Vincentelli, and A. Wang, "Technology Mapping in MIS," *Proc. IEEE Int'l Conf. on Computer-Aided Design*, pp. 116-119, 1987.

[5] Francis, R. J., J. Rose, and K. Chung, "Chortle: A Technology Mapping Program for Lookup Table-Based Field Programmable Gate Arrays," *Proc. 27th ACM/IEEE Design Automation Conference*, pp. 613-619, 1990.

[6] Francis, R. J., J. Rose, and Z. Vranesic, "Technology Mapping of Lookup Table-Based FPGAs for Performance," *Proc. Int'l Conf. Computer-Aided Design*, pp. 568-571, Nov., 1991.

[7] Francis, R. J., J. Rose, and Z. Vranesic, "Chortle-crf: Fast Technology Mapping for Lookup Table-Based FPGAs," *Proc. 28th ACM/IEEE Design Automation Conference*, pp. 613-619, 1991.

[8] Gabow, H., "An Efficient Implementation of Edmonds' Algorithm for Maximum Matching on Graphs," *Journal of the ACM*, Vol. 23, pp. 221-234, Apr. 1976.

[9] Hoover, H. J., M. M. Klawe, and N. J. Pippenger, "Bounding Fan-out in Logic Networks," *Journal of the ACM*, Vol. 31, pp. 13-18, Jan. 1984.

[10] Huffman, D. A., "A method for the construction of minimum redundancy codes," *Proc. IRE 40*, pp. 1098-1101, 1952.

[11] Karplus, K., "Xmap: A Technology Mapper for Table-lookup Field-Programmable Gate Arrays," *Proc. 28th ACM/IEEE Design Automation Conference*, pp. 240-243, 1991.

[12] Keutzer, K., "DAGON: Technology Binding and Local Optimization by DAG Matching," *Proc. 24th ACM/IEEE Design Automation Conference*, pp. 341-347, 1987.

[13] Lawler, E. L., K. N. Levitt, and J. Turner, "Module Clustering to Minimize Delay in Digital Networks," *IEEE Transactions on Computers*, Vol. C-18(1) pp. 47-57, January 1969.

[14] Murgai, R., et al, "Logic Synthesis Algorithms for Programmable Gate Arrays," *Proc. 27th ACM/IEEE Design Automation Conf.*, pp. 620-625, 1990.

[15] Murgai, R., N. Shenoy, R. K. Brayton, and A. Sangiovanni-Vincentelli, "Performance Directed Synthesis for Table Look Up Programmable Gate Arrays," *Proc. Int'l Conf. Computer-Aided Design*, pp. 572-575, Nov., 1991.

[16] Murgai, R., N. Shenoy, R. K. Brayton, and A. Sangiovanni-Vincentelli, "Improved Logic Synthesis Algorithms for Table Look Up Architectures ," *Proc. Int'l Conf. Computer-Aided Design*, pp. 564-567, Nov., 1991.

[17] Roth, J. P. and R. M. Karp, "Minimization Over Boolean Graphs," *IBM Journal of Research and Development*, pp. 227-238, April 1962.

[18] Woo, N.-S., "A Heuristic Method for FPGA Technology Mapping Based on the Edge Visibility," *Proc. 28th ACM/IEEE Design Automation Conference*, pp. 248-251, 1991.

[19] Xilinx, *The Programmable Gate Array Data Book*, Xilinx, San Jose (1989).