Technology Mapping for Large Complex PLDs

Jason Helge Anderson and Stephen Dean Brown
Department of Electrical and Computer Engineering
University of Toronto
10 King's College Road
Toronto, Ontario, Canada M5S 3G4
{janders|brown}@eecg.toronto.edu

1. ABSTRACT

In this paper we present a new technology mapping algorithm for use with complex PLDs (CPLDs), which consist of a large number of PLA-style logic blocks. Although the traditional synthesis approach for such devices uses two-level minimization, the complexity of recently-produced CPLDs has resulted in a trend toward multi-level synthesis. We describe an approach that allows existing multi-level synthesis techniques [13] to be adapted to produce circuits that are well-suited for implementation in CPLDs. Our algorithm produces circuits that require up to 90% fewer logic blocks than the circuits produced by a recently-published algorithm.

1.1 Keywords

PLA-style logic blocks, programmable logic devices, technology mapping.

2. INTRODUCTION

Field-programmable gate arrays (FPGAs) and complex programmable logic devices (CPLDs) have become widely used for the implementation of digital circuits, with each type of chip representing about half of a two billion dollar industry. FPGAs and CPLDs are user-programmable chips that consist of an array of logic blocks and a configurable interconnection network. Most FPGAs have logic blocks based on look-up-tables (LUTs) [14][2], and some have multiplexer-based logic blocks [1]. CPLDs have PLA-style logic blocks [3][2].

In recent years, synthesis for look-up-table-based FPGAs has been an active area of research and development. Many LUT-based technology mapping algorithms have been proposed. Some of these algorithms focus on minimizing either area [8][9] or depth [4][10], while others allow a user to explore the area/depth trade-off [5]. By contrast, very little has been published on synthesis for CPLDs. Several CAD tool vendors offer products that can perform technology mapping for CPLDs, but the algorithms are proprietary and are not publicized. This paper describes a new CAD tool, called *TEMPLA*, that performs technology mapping for devices with PLA-style logic blocks.

The goal of our technology mapping algorithm is to minimize the number of logic blocks required to implement the resulting circuits, where each logic block has the structure shown in Figure 1. The block is characterized by the tuple (*I, P, O*) where the parameters *I, P,* and *O* represent the number of block inputs, product terms, and outputs, respectively. Each output of the block has an associated flip-flop which can either be used or by-passed. Each ✖ in the figure represents a programmable switch. The block in the figure can be adapted to represent the logic blocks in most commercially-available CPLDs. As an example, in Section 4.4 we discuss using our approach to technology mapping for the CPLDs available from Vantis (AMD) [3].

This paper is organized as follows: Background and related work is discussed in Section 3. We present our technology mapping algorithm in Section 4. The results of a comparative experimental study are given in Section 5. Final remarks and suggestions for future work are provided in Section 6.

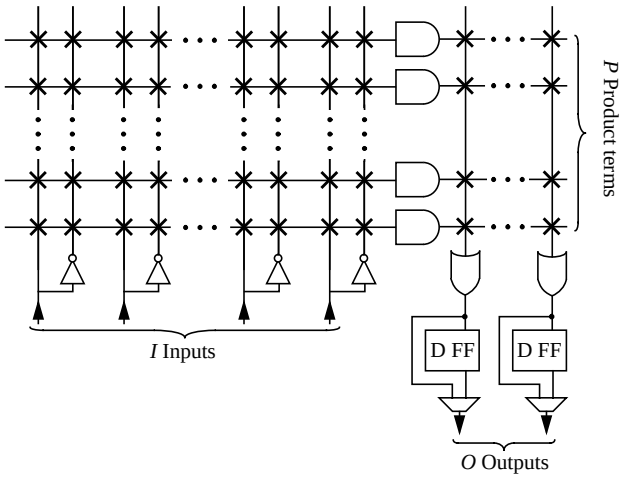

Figure 1. PLA-Style Logic Block.

3. BACKGROUND AND RELATED WORK

The combinational part of a digital circuit can be represented by a directed acyclic graph (DAG). Each node in the DAG represents a single logic function in the circuit; the edges in the DAG represent dependencies between logic functions.

1-58113-049-x-98/0006/$3.50

The logic function associated with a node in a DAG can be represented in sum-of-products (SOP) form.

PLA-style blocks are referred to as coarse-grained logic blocks because they typically have a large number of inputs, and hence can realize a large number of different logic functions. Since a PLA-style block contains a programmable AND-array followed by an OR-array, the traditional approach for synthesis of circuits for implementation in such blocks uses two-level minimization. This means that the logic functions in a circuit are collapsed into sum-of-products form. Modern CPLDs contain a large number of PLA-style blocks, with currently-available devices containing up to about 100 blocks. Such large chips allow the implementation of complex circuits for which two-level minimization is not efficient. Hence, the recent trend for synthesis when targeting CPLDs is to make use of multi-level synthesis.

Our approach to synthesis of circuits for realization in PLA-style blocks represents a combination of techniques. We use multi-level logic optimization and traditional technology mapping as a first step. Then, since it is not feasible to directly use traditional technology mapping for PLA-based architectures, as discussed below, we *partially-collapse* the resulting circuit to make efficient use of PLA-style blocks. We describe our algorithm in detail in the next section.

In traditional technology mapping, a library is used to specify all of the logic functions that can be used in the resulting circuit. This approach is not feasible for PLA-based architectures, because the library would be too large to be repetitively searched during technology mapping. One of the few publicized approaches for mapping circuits into PLA-style blocks is found in [11]. It adapts a LUT-based technology mapper and sets the number of LUT inputs to the number of PLA-style block inputs. Then, any node containing more product terms than allowable in the PLA-style block is decomposed into smaller nodes. Finally, the nodes are packed into the multi-output PLA-style blocks.

4. TEMPLA: A TECHNOLOGY MAPPING ALGORITHM FOR LARGE COMPLEX PLDS

In TEMPLA, the technology mapping problem is broken into three phases: performing an optimal tree mapping, heuristic partial collapsing, and bin packing. This algorithmic flow is similar to that of the Chortle-crf technology mapper [9] for LUT-based architectures.

4.1 Phase I: Optimal Tree Mapping

Technology mapping begins by partitioning a circuit's DAG into a forest of fanout-free trees. This is accomplished by identifying the nodes within the DAG that have an out-degree greater than one, and using these nodes as 'breaking points', as illustrated in Figure 2. The reason for this step is to divide the technology mapping problem into smaller and simpler sub-problems. Technology mapping for fanout-free trees is simpler because no node in a fanout-free tree has an out-degree greater than one and, therefore, it is not necessary to consider replication of logic at this juncture of the algorithm.

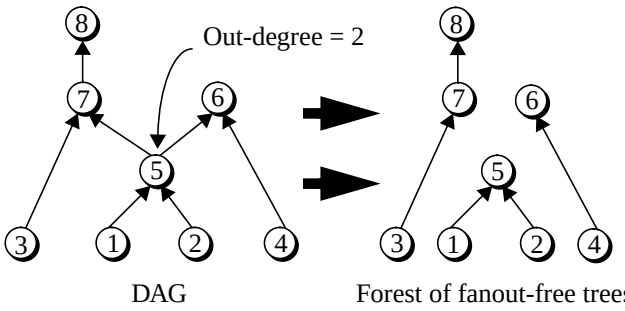

Figure 2. Partitioning a DAG into a forest of fanout-free trees.

Primary input nodes are added to each of the fanout-free trees by modifying them in the following way: for each leaf vertex, n, in a fanout-free tree, $T = (V, E)$, a new primary input node, p, is added to the vertex set V. An edge, $e = (p, n)$, is created and added to the edge set E. The primary input node p is a dummy node and implements no logic function. Before explaining the algorithm further, we must define several terms:

feasible node - a node implementing a single combinational logic function with the property that when simplified, it has less than or equal to I inputs and less than or equal to P product terms. We use the two-level logic minimization provided by Espresso [7] to simplify combinational nodes.

feasible subtree - a subtree of a fanout-free tree with the property that it can be collapsed into a single feasible node. Feasible subtrees are not allowed to possess any of the dummy primary input nodes.

cone at n - a subtree of a fanout-free tree consisting of a node, n, and all of n's predecessors.

size of node n - the size of a node with p product terms and i inputs is equal to $p \times i$.

After the partitioning of the DAG, dynamic programming [6] is used to map each fanout-free tree into a new tree possessing the minimum number of feasible nodes. The trees in the forest can be mapped in any order.

To map a fanout-free tree, $T = (V, E)$, it is traversed in a bottom-up (leaves to root) manner. As each node, n, is visited in turn, the algorithm proceeds to find the set, $S(n)$, of all feasible subtrees of T rooted at n. A cost is computed for each feasible subtree and the feasible subtree of minimum cost is selected and stored at node n. $Cost(n)$ is an integer that refers to this minimum cost.

Primary input nodes implement no logic function and are assigned a cost of zero. All other nodes in V initially have no cost assigned. Three steps are performed repetitively until all of the nodes in V have been assigned a cost.

Step 1: Select a node, n, from V that has not yet been assigned a cost but whose fan-in nodes have been assigned a cost (this implies a bottom-up tree traversal).

Step 2: Determine $S(n)$ - the set of all feasible subtrees rooted at n.

Step 3: Assign a cost to node n using the expression:

$$Cost(n) = min_{T' \in S(n)}\left(1 + \sum_{u \in FI(T')} Cost(u)\right) \quad (1)$$

where $T' = (V', E')$ is a feasible subtree rooted at n belonging to the set $S(n)$; $FI(T')$ is the set of nodes in the fanout-free tree, $T = (V, E)$, that are not nodes in the feasible subtree T' but that are inputs to nodes in T'. More formally:

$$FI(T') = \{v | v \in V, v \notin V', (v, w) \in E, w \in V'\} \quad (2)$$

In Equation (1), $Cost(n)$ represents the minimum number of feasible nodes needed to implement the cone at n. Each subtree, T', in $S(n)$ can be collapsed into one feasible node in the mapping solution; this is the reason for the 1 inside the brackets of Equation (1). The summation term tallys the costs of nodes in T that are inputs to nodes in the subtree T'. The min function selects the feasible subtree rooted at n that results in the minimum-cost mapping of the cone at n. Figure 3 shows a node, n, along with three feasible subtrees rooted at n. The cost of each of n's predecessors is shown internal to each node. The last node to be assigned a cost is the root of the fanout-free tree being mapped.

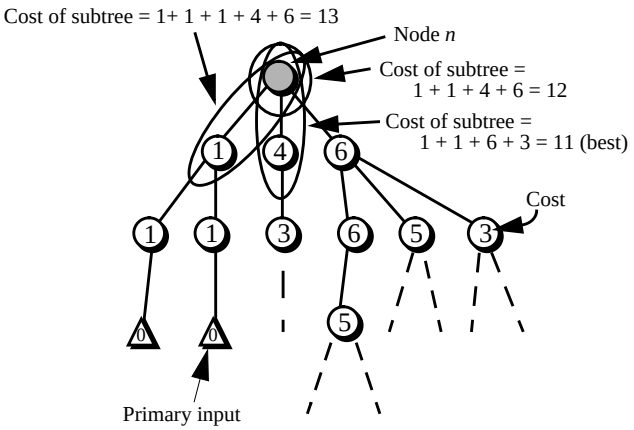

Figure 3. Computation of feasible subtree cost.

Steps 1 to 3 are repeated until all of the nodes have been assigned a cost, at which point the final mapping solution for the tree is specified according to the minimum-cost feasible subtree stored at each node. The final solution is extracted by first considering the root, r, of the original fanout-free tree. The minimum-cost feasible subtree stored at the root is implemented as a new feasible node in the mapping solution. Nodes in T that are inputs to this new feasible node are then added to a node set, M. Mapping proceeds by removing a node, m, from M, implementing the subtree stored at m as a new node in the mapping

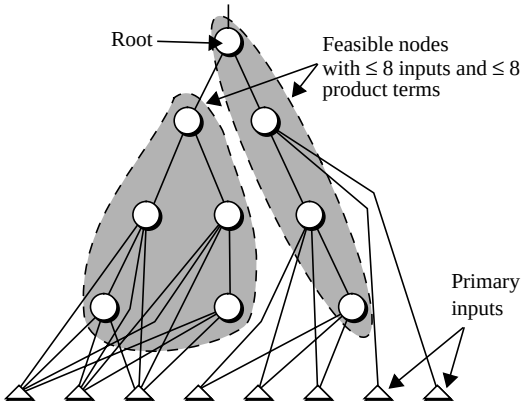

Figure 4. Mapping solution for a tree in MCNC circuit s5378 for *I*=8, *P*=8.

solution, and lastly, identifying nodes in T that are inputs to the newly created node and adding them to the set M. This process continues until M contains only primary input nodes at which time a network with the minimum number of feasible nodes has been created to implement the function of the original tree. An example of the mapping solution produced by phase I of TEMPLA for a tree in a real circuit is shown in Figure 4.

One aspect of Step 2 of the algorithm has to be mentioned. A recursive algorithm is used to find $S(n)$. The constraint that a feasible subtree must have a limited number of product terms adds substantial complexity to the problem. For example, consider the case of finding the set of feasible subtrees for a node, n, with two fanin nodes, A and B. Assume that the subtree consisting of n and A is a feasible subtree but that the subtree consisting of n and B is not feasible because it has more than P product terms. Complexity is introduced because the infeasibility of the n and B subtree does not imply the infeasibility of the n and A and B subtree. Specifically, the n and A and B subtree may be feasible because the subtree consisting of n and A may simplify into a feasible node containing fewer product terms than were originally in node n. If we compare this problem to technology mapping for look-up-tables, for which only the number of inputs to the block needs to be considered, it is clear that technology mapping for PLA-style blocks has added complexity.

After performing phase I on all of the fanout-free trees within a circuit's DAG, the mapping solutions for each tree are put back together into a complete circuit. The algorithm then considers replacing each node in the circuit with its complemented form. This can be beneficial if the complemented form of a node contains fewer product terms than its uncomplemented form, and therefore, this optimization is considered after each of our algorithm's three phases. Phase II attempts to reduce the number of nodes in the circuit by collapsing nodes across tree boundaries.

4.2 Phase II: Heuristic Partial Collapsing

Any node that can be collapsed into all of its fanout nodes

can be eliminated, provided that all nodes remain feasible after the collapsing. This introduces another optimization problem since collapsing some nodes into their fanout nodes may preclude the possibility of collapsing other nodes into their fanout nodes. Several criteria were studied empirically using 30 benchmark circuits to determine how best to choose nodes to collapse. A total of 19 of the 30 circuits used in this study are large MCNC benchmarks [15], 10 circuits are HDL specifications developed by the authors, and one circuit is a processor benchmark from the PREP synthesis suite [12]. The criteria listed below refer to nodes *to be collapsed* into their fanout nodes, and not the new node(s) that would exist after collapsing is done. The criteria considered are:

1. Inputs - prefer to collapse nodes with fewer inputs.

2. Product terms - prefer to collapse nodes with fewer product terms.

3. Node size - prefer to collapse small nodes.

4. Fanout - prefer to collapse nodes with low fanout.

To evaluate the criteria, each was applied individually as the selection criteria for partial collapsing. The number of circuit nodes before and after collapsing was determined and a percentage reduction was computed for each benchmark circuit. These percentages were then averaged; hence, each circuit was treated equally in the comparison. The results of this experiment showed that each of the criteria performed similarly, on average. We decided to use node size as the primary criteria for selecting nodes to collapse, with fanout used as a secondary criteria.

Since phase II uses a heuristic approach when choosing nodes to collapse, we attempted to evaluate the quality of our solution. To do this, we implemented an exhaustive algorithm based on the branch-and-bound method to perform the equivalent of phases I and II. Although the exhaustive approach could be used only for small circuits, due to its time-complexity, it revealed an interesting property of the problem being solved. Recall that the PLA-style blocks may have multiple outputs. For such blocks, comparing our heuristic solution to the exhaustive solution revealed that it is sometimes best to limit the size of nodes created during phase II to better facilitate the packing of nodes into multi-output blocks. TEMPLA was modified to deal with this issue as discussed below.

When a node is collapsed into its fanout nodes, the sum of the sizes of the resultant nodes after collapsing may be larger than the sum of the sizes of nodes before collapsing. The TEMPLA algorithm allows a user to limit this size increase by varying the parameter β in the following relation:

$$\sum_{t \in T} size(t) \leq \beta \cdot \left(size(v) + \sum_{s \in S} size(s) \right) \qquad (3)$$

where v is the node to be collapsed into its fanout nodes; S is the set of v's fanout nodes before any collapsing; and T is the set of v's fanout nodes after v has been collapsed into them. The algorithm will not collapse a node into its fanout nodes if Relation (3) evaluates false.

When the logic blocks in the target architecture have only one output, β should be set to a large number, which places no restrictions on the size of nodes after collapsing. However, for multi-output logic blocks, it may be advantageous to set β to a smaller value. This was investigated experimentally in the context of the third phase of our algorithm, and the results are shown in the next section.

4.3 Phase III: Bin Packing

The final phase of TEMPLA packs circuit nodes into the multi-output blocks available in the target architecture. This is accomplished using a first-fit-decreasing bin packing algorithm that attempts to maximize the number of shared inputs between nodes that are packed into the same logic block. The bin packing algorithm used is described using pseudo code in Figure 5. Single input nodes that implement inverters are ignored during bin packing because signals can be inverted for 'free' in PLA-style logic blocks, since each input to the block is available in both true and complemented form.

To investigate what value of β in Relation (3) is appropriate for multi-output blocks, β was varied while 30 benchmark circuits were mapped into logic blocks with the parameters (10, 12, 4). The number of blocks needed to implement each circuit was compared to that attained when β was set to a large value (unrestricted collapsing) and a percentage decrease in the number of logic blocks was computed. The average percentage decrease over all 30 circuits is shown in Table 1.

The results in Table 1 suggest that it is not beneficial to pack as much logic as possible into each feasible node before packing the nodes into multi-output logic blocks. Setting β equal to 1.5 is the best choice when targeting blocks with four outputs.

Table 1: Effect of restricted partial collapsing.

β	**Average % decrease in # of logic blocks**
1.0	-1.4
1.25	3.7
1.5	4.3
1.75	3.6
2.0	2.9
2.25	3.0

4.4 Adapting TEMPLA to Target the Vantis (AMD) Mach 4 [3]

As an example of how TEMPLA can be applied to a commercial CPLD product, this section describes its use for the Vantis (AMD) Mach 4 CPLDs [3]. Each logic block in

the Mach 4 has 33 inputs, 90 product terms, and 16 outputs. Eighty of the 90 product terms are grouped into 16 clusters of five and these product terms are available to implement combinational logic. Each of the 16 'OR gates' in the logic block is allocated a cluster of five product terms; however, clusters may be redirected from an OR gate to another adjacent OR gate (leaving one OR gate unused). A maximum of 20 product terms are allowed to feed a single OR gate.

Phases I and II of TEMPLA can be used to create nodes that pack efficiently into Mach 4 logic blocks by setting *I* equal to 33 and *P* equal to 20. These first two phases could be augmented with cost functions to reflect the notion that a node with between one and five product terms costs less to implement than a node with between six and ten product terms, since a product term cluster would need to be redirected in the latter case. The cost of a node would be proportional to the number of product term clusters that need to be redirected to implement it. Nodes could be packed into the Mach 4 logic blocks using a slightly modified version of phase III of TEMPLA. Hence, we believe that relatively few changes to our algorithm would be required for it to efficiently target the Mach 4 architecture.

5. EXPERIMENTAL RESULTS

Our algorithm has been implemented in the C language within the SIS [13] framework, allowing it to access the I/O routines and two-level logic minimization algorithms within SIS. To assess the quality of mapping solutions produced by our tool, it was compared with the approach used in [11], called DDMAP. The first step of DDMAP is to apply a look-up-table technology mapper; we used Level-Map [8], which minimizes the number of look-up-tables needed, to perform this initial mapping.

Table 2 shows the results when the technology mappers are used to map circuits into PLA-style blocks with the parameters (10, 12, 4). These parameters were chosen based on previous research in [11], which showed that logic blocks with 8-10 inputs, 12-13 product terms, and 3-4 outputs are the most area-efficient PLA-style logic blocks. The results for 15 of the 30 circuits used in this study are shown in the table. Before applying TEMPLA or DDMAP, each of the circuits was synthesized using the Synopsys Design Compiler. This resulted in an optimized multi-level netlist of gates from an intermediate library. The intermediate library consists of elements suitable for mapping circuits into PLA-based architectures. The first column in the table lists the name of each benchmark circuit. The second lists the number of logic blocks needed to implement each circuit when TEMPLA is used. The third column shows the running time in seconds for TEMPLA on a 300 MHz SPARCstation. The fourth column gives the results for DDMAP; a percentage is given in brackets which represents the amount of additional logic blocks needed to implement each circuit in comparison with TEMPLA. On average, when the 15 circuits shown in Table 2 are mapped using DDMAP, they require 90% more blocks than when TEMPLA is used. Over all 30 circuits considered, the DDMAP solutions contained 93.8% more blocks on average.

Notice that TEMPLA performs poorly for the benchmark 'ex5p'. For this circuit, DDMAP produces a solution with nearly 80% fewer blocks than TEMPLA. The ex5p circuit is a combinational circuit possessing 8 primary inputs, and 63 primary outputs. Since the number of inputs to the circuit is less than the number of inputs to the logic blocks in the (10, 12, 4) architecture, Level-Map produces a mapping containing 63 nodes: one node for each primary output. Level-Map produces such a mapping because it is able to deal effectively with reconvergent paths within circuits. Furthermore, for this circuit, most of the nodes in the Level-Map solution happen to be feasible nodes. Many of the nodes have common inputs, allowing several nodes to be packed into each 4-output logic block. To verify that the exploitation of reconvergent paths was the reason for the superior mapping, the circuit was mapped with the LUT-based technology mapper, Chortle-crf [9], which deals with reconvergence in only a limited way. Chortle-crf produced a mapping containing 363 nodes which is significantly greater than the 63 nodes in the Level-Map solution. Since TEMPLA breaks up a circuit into fanout-free trees, it is not able to exploit reconvergent paths effectively, and hence,

```
nodeSet ← Set of all nodes in network (minus single input inverters)
while (nodeSet is not empty) {
        plaBlock ← empty block /* allocate a new PLA-style logic block */
        nodeSel ← largest node in nodeSet (node size = number of inputs x number of product terms)
        Add nodeSel to plaBlock
        Remove nodeSel from nodeSet
        while (nodeSet is not empty and there are nodes in nodeSet that can fit into plaBlock) {
                nodeSel ← node from nodeSet that has the largest number of inputs in common with the nodes
                        already in plaBlock; the node must be able to fit into plaBlock; use node size to break ties
                Add nodeSel to plaBlock
                Remove nodeSel from nodeSet
                }
        }
```

Figure 5. Maximum shared input bin packing algorithm.

produces an inferior solution for this circuit.

6. FINAL REMARKS AND FUTURE WORK

In this paper we presented a new technology mapping CAD tool for CPLDs with a large number of PLA-style logic blocks. Our algorithm breaks the technology mapping problem into three phases: optimal tree mapping, heuristic partial collapsing, and bin packing. The experimental study described in the previous section shows that our tool produces mapping solutions containing fewer logic blocks than solutions produced by a recently-published technique.

One direction for future work is to modify our algorithm to target a specific commercially available CPLD, as discussed in Section 4.4, and compare the mapping results to those produced by existing commercial CAD tools. Since development of CPLDs that contain sufficient numbers of PLA-style blocks to benefit from multi-level synthesis has only recently come about, we believe that synthesis issues for such chips is an important area for future research.

Table 2: Experimental results for (10, 12, 4) architecture.

Circuit	TEMPLA (# blocks)	TEMPLA running time (seconds)	DDMAP [11] (# blocks (% more))
alu4	155	29.3	199 (28.4)
apex4	193	30.2	193 (0.0)
clma	957	815.4	1458 (52.4)
cps	120	18.3	159 (32.5)
dalu	64	9.0	102 (59.4)
ex5p	132	18.8	27 (-79.5)
misex3	154	27.8	214 (39.0)
pdc	618	281.6	1221 (97.6)
s38417	603	495.2	1208 (100.3)
seq	229	52.8	337 (47.2)
fir	249	123.3	1424 (471.9)
fsm8_8_13	49	5.5	58 (18.4)
pmac	237	126.6	911 (284.4)
psdes	151	37.0	301 (99.3)
sort	138	29.8	275 (99.3)
			Average over 15 circuits above: 90%
			Average over all 30 circuits: 93.8%

7. ACKNOWLEDGMENTS

The authors gratefully acknowledge the financial support of Chip Express Corporation and the Government of Ontario. Thanks to Dr. Jack Kouloheris for providing code for the DDMAP [11] algorithm.

8. REFERENCES

[1] *ACT 1 Series FPGAs Data Sheet*, Actel Corporation, 1996.

[2] *The Altera Data Book*, Altera Corporation, 1996.

[3] *The MACH 4 Family Data Sheet*, Advanced Micro Devices, 1996.

[4] J. Cong and Y. Ding, "FlowMap: An Optimal Technology Mapping Algorithm for Delay Optimization in Lookup-Table Based FPGA Designs", *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, Vol. 13, No. 1, January 1994, pp. 1-11.

[5] J. Cong and Y. Hwang, "Simultaneous Depth and Area Minimization in LUT-based FPGA Mapping", *UCLA Department of Computer Science Technical Report*, CSD TR-9500001.

[6] T. H. Cormen, C. E. Leiserson and R. L. Rivest, *Introduction to Algorithms*, McGraw-Hill Book Company, Toronto, 1994.

[7] Giovanni De Micheli, *Synthesis and Optimization of Digital Circuits*, McGraw-Hill Inc., Toronto, 1994.

[8] A. H. Farrahi and M. Sarrafzadeh, "Complexity of the Lookup-Table Minimization Problem for FPGA Technology Mapping", *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, Vol. 13, No. 11, November 1994, pp. 1319-1332.

[9] R.J Francis, J. Rose and Z. Vranesic, "Chortle-crf: Fast Technology Mapping for Lookup Table-Based FPGAs", *28th ACM/IEEE Design Automation Conference*, June 1991, pp. 227-233.

[10] R. J. Francis, J. Rose and Z. Vranesic, "Technology Mapping of Lookup Table-Based FPGAs for Performance", *1991 IEEE Conference on Computer-Aided Design*, pp. 568-571.

[11] J. L. Kouloheris, "Empirical Study of the Effect of Cell Granularity on FPGA Density and Performance", *Ph.D. Thesis*, Department of Electrical Engineering, Stanford University, 1993.

[12] *Programmable Electronics Performance Corporation Test Benches*, http://www.prep.org, 1996.

[13] E. M. Sentovice et al., "SIS: A System for Sequential Circuit Synthesis", *Technical Report UCB/ERL M92/41*, Electronics Research Laboratory, Department of Electrical Engineering and Computer Science, University of California, Berkeley, 1992.

[14] *The Programmable Logic Data Book*, Xilinx Corporation, 1994.

[15] S. Yang, "Logic Synthesis and Optimization Benchmarks", *Technical Report*, Microelectronics Center of North Carolina, 1991.