Short Papers

Performance-Driven Mapping for CPLD Architectures

Deming Chen, Jason Cong, Milos Ercegovac, and Zhijun Huang

***Abstract*—We present a performance-driven programmable logic array mapping algorithm (PLAmap) for complex programmable logic device architectures consisting of a large number of PLA-style logic cells. The primary objective of the algorithm is to minimize the depth of the mapped circuit. We also develop several techniques for area reduction, including threshold control of PLA fanouts and product terms, slack-time relaxation, and PLA packing. We compare PLAmap with a previous algorithm TEMPLA (Anderson and Brown 1998) and a commercial tool Altera Multiple Array MatriX (MAX) + PLUS II (Altera Corporation 2000) using Microelectronics Center of North Carolina (MCNC) benchmark circuits. With a relatively small area overhead, PLAmap reduces circuit depth by 50% compared to TEMPLA and reduces circuit delay by 48% compared to MAX + PLUS II v9.6.**

***Index Terms*—Complex programmable logic device (CPLD) architecture, technology mapping.**

I. Introduction

Programmable logic devices (PLDs) have been widely used for digital system implementation due to their fast manufacturing turnaround time, low startup costs, and ease of design changes. There are two major types of PLDs: field programmable gate arrays (FPGAs) and complex programmable logic devices (CPLDs). The logic cells in FPGAs are usually fine-grained programmable blocks that produce high logic densities and much design flexibility. However, the interconnect structures for FPGAs are complex and the delay is often not predictable in prelayout stages. In contrast, the logic cells in CPLDs are coarse-grained two-level AND-OR programmable logic arrays (PLAs), which are also known as *product-term (p-term) blocks*. Although PLAs have lower logic densities, their interconnect structures are much simpler and the delay is more predictable.

Technology mapping is the first device-dependent step in implementing a circuit design on PLDs. In contrast with extensive studies on technology mapping for FPGAs, limited work has been done on mapping for CPLDs. There is almost no significant research done from the perspective of performance-driven CPLD mapping. Hasan *et al.* [8] proposed a fast heuristic partition method for PLA-based structures. Kouloheris presented DDMap [11] which adapted a lookup table (LUT)-based technology mapper and set the number of LUT inputs to the number of PLA-style block inputs. Any node containing more product-terms than allowable in the PLA was then decomposed into smaller nodes. Finally, the nodes were packed into multioutput PLA-style blocks. Liu *et al.* [13] addressed the problem of partitioning a large PLA into a number of smaller sub-PLAs such that the total area of these sub-PLAs and the cycle time of the partitioned circuit were minimized. Anderson and Brown [1] developed TEMPLA with the goal of minimizing the number of PLAs required to implement circuits on CPLDs. The algorithmic flow of TEMPLA included three phases: optimal tree mapping, heuristic partial collapsing, and bin packing, which were similar to that of the Chortle-crf technology mapper [7] for LUT-based FPGAs. Kania [9] proposed a mapping algorithm using multioutput function graphs for PAL-based devices. Kim *et al.* [10] developed a CPLD mapping algorithm for area minimization under the time constraint. In k_m_flow [6], Cong *et al.* emphasized that it was inherently difficult to map logic into multioutput PLA-style programmable cells. Rather than targeting multioutput PLAs, k_m_flow is a technology mapper for single-output macrocells with k input and m p-terms.

In this paper, we present a performance-driven mapping algorithm named *PLAmap* for CPLDs based on multioutput PLAs. Each PLA has the structure shown in Fig. 1. A (k, m, p)-PLA has k inputs, m p-terms and p outputs. Both small PLAs such as (10,12,4)-PLAs studied in [11] and large commercial PLAs such as Altera (36,80,16)-PLAs are considered. The basic algorithm is adjusted for different PLA granularity because the number of p-terms often increases exponentially with the number of the circuit inputs [14]. In addition, structural constraints in commercial CPLDs are taken into account. Our algorithm differs from previous CPLD mapping algorithms in three aspects. First, it is a performance-driven mapping algorithm for general CPLD structures. Second, the mapping step directly generates multioutput PLAs while traditional algorithms only produce multioutput PLAs in the final packing step. Third, applications to commercial CPLD structures are considered. The rest of this paper is organized as follows. Section II defines terminology and formulates the problem. Section III describes the algorithm in detail. Section IV gives the experimental results and Section V concludes this work.

II. Definitions and Problem Formulation

A Boolean network can be represented as a directed acyclic graph (DAG) in which each node represents a logic gate, and a directed edge (i, j) exists 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. A *predecessor* of node v is any node u such that there is a directed path from u to v. In this case, node v is a *successor* of u. We use *input* (v) to denote the set of nodes that supplies inputs to node v.

A *cluster* rooted at a node set $\mathbf{R}$, denoted as $\mathrm{CST}_{\mathbf{R}}$, is a subgraph of the Boolean network such that any path connecting two arbitrary nodes in $\mathrm{CST}_{\mathbf{R}}$ lies entirely in $\mathrm{CST}_{\mathbf{R}}$. output$(\mathrm{CST}_{\mathbf{R}})$ is also used to represent the root set $\mathbf{R}$ since these roots are also the outputs of $\mathrm{CST}_{\mathbf{R}}$. node$(\mathrm{CST}_{\mathbf{R}})$ represents the set of nodes contained in $\mathrm{CST}_{\mathbf{R}}$. input$(\mathrm{CST}_{\mathbf{R}})$ denotes the set of distinct nodes outside of $\mathrm{CST}_{\mathbf{R}}$ that supply inputs to the nodes in node$(\mathrm{CST}_{\mathbf{R}})$. A *subcluster* $\mathrm{CST}_{\mathbf{T}}$ of $\mathrm{CST}_{\mathbf{R}}$ is a cluster that is rooted at set $\mathbf{T}$ and is completely contained in $\mathrm{CST}_{\mathbf{R}}$. In this case, $\mathrm{CST}_{\mathbf{R}}$ is called the *supercluster* of $\mathrm{CST}_{\mathbf{T}}$. If $\mathbf{R}$ contains only one node v (i.e., $|\mathbf{R}| = 1$), $\mathrm{CST}_{\mathbf{R}}$ represents a single-output network rooted at node v. In this special case, $\mathrm{CST}_{\mathbf{R}}$ can be simply denoted as CST_v. In general, $\mathrm{CST}_{\mathbf{R}}$ corresponds to a multioutput network.

A $\mathrm{CST}_{\mathbf{R}}$ can be optimized for the PLA structure shown in Fig. 1. The set of p-terms of the optimized PLA is defined as the set of p-terms of $\mathrm{CST}_{\mathbf{R}}$, denoted as pterm$(\mathrm{CST}_{\mathbf{R}})$. A $\mathrm{CST}_{\mathbf{R}}$ is said to be (k, m, p)*-feasible* if and only if $|\text{input}(\mathrm{CST}_{\mathbf{R}})| \le k$,

Manuscript received March 27, 2002; revised December 27, 2002. This work was supported in part by the Altera Corporation and the Lattice Semiconductor Corporation under the California MICRO program and in part by the National Science Foundation under Grant MIP-9357582, Grant MIP-9725771, and Grant CCR-0096383. This paper was recommended by Associate Editor M. Pedram.

D. Chen, J. Cong, and M. Ercegovac are with the Computer Science Department, University of California, Los Angeles, CA 90024 USA (e-mail: demingc@cs.ucla.edu).

Z. Huang was with the Computer Science Department, University of California, Los Angeles, CA 90024 USA. He is now with Synopsys Inc., Hillsboro, OR 97124 USA (e-mail: zjhuang@synopsis.com).

Digital Object Identifier 10.1109/TCAD.2003.818120

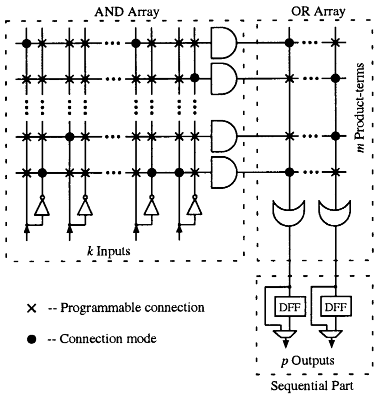

Fig. 1. (k, m, p)-PLA structure.

$|\text{pterm}(\text{CST}_{\mathbf{R}})| \le m$ and $|\text{output}(\text{CST}_{\mathbf{R}})| \le p$ are all satisfied. Otherwise, it is (k, m, p)*-infeasible*. The technology mapping problem for CPLDs is to cover a given Boolean network with (k, m, p)*-feasible* clusters, which are then converted to PLAs. A mapping solution S is a DAG where each node of the DAG is (k, m, p)*-feasible,* and a directed edge exists for each direct connection from any output$(\text{CST}_{\mathbf{R1}})$ to any input$(\text{CST}_{\mathbf{R2}})$.

Two factors determine the delay of a CPLD circuit: delay in p-term blocks and delay in interconnection paths. Since layout information is not available at the mapping stage, we assume that each interconnection edge contributes a constant delay, which is reasonable in CPLD structures. If each cluster is feasible and transformed into a PLA, we can simply approximate the circuit delay by using a *unit PLA-delay model*. A *unit PLA-delay* is defined as the delay of the AND-OR path in a PLA. Each PLA along the critical path contributes one unit PLA-delay to the logic depth of the network. Our main objective is to compute a performance-driven mapping solution that minimizes the logic depth. The second objective is to reduce the area measured in terms of the number of PLAs without sacrificing the performance. A threshold control technique is also proposed for area/delay tradeoff.

III. Algorithm Description

A. Overview

Our algorithm, PLAmap, consists of three stages: first, label the network from PIs to POs; second, map the labeled network into (k, m, p)-PLAs from POs to PIs; and third, pack PLAs to further reduce the area. This algorithm flow is similar to that of DAG-Map [2] for LUT-based FPGAs. We assume that the input network has already been decomposed into a *two-bounded* network. Several good decomposition algorithms, such as tech_decomp from sequential interactive synthesis (SIS) [15] and *dmig* from DAG-Map [2], can be used for our purpose. Actually, as long as each node in the input network is (k, m, p)-feasible, the network can be directly accepted by PLAmap. The *two-bounded* network is chosen to ensure the same starting point for every input network. In addition, smaller gates are more easily packed for delay optimization [5].

B. Labeling Stage

The labeling stage computes the mapping depth and provides clustering information for the subsequent mapping stage. To minimize depth in the final PLA network, we first label the Boolean network targeting single-output $(k, m, 1)$-PLAs so that we can form a PLA cluster as deep as possible. In the mapping and packing stages, we then generate multioutput (k, m, p)-PLAs ($p > 1$).

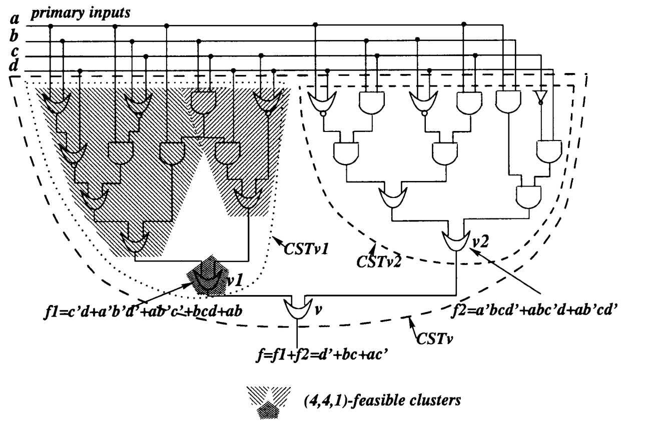

Fig. 2. Nonmonotone properties in CPLD mapping.

```
/* Stage 1: labeling the network */
for each PI node $v$ do
    $label(v) = 1$;
end for;
$T$ = list of non-PI nodes in a topological order;
while $T$ is not empty do
    remove the first node $v$ from $T$;
    $l = max\{label(u)|u \in input(v)\}$;
    form $CST_v$ with $v$'s label-$l$ predecessors;
    if $CST_v$ is $(k, m, 1)$-feasible then
        $label(v) = l$;
    else $label(v) = l + 1$;
    end if;
end while;
```

Fig. 3. Labeling procedure of PLAmap.

Both DAG-Map [2] and FlowMap [3] use labeling techniques as their first step to compute the best mapping depth. FlowMap offers a polynomial time algorithm to find the optimum depth for LUT-based FPGA mapping. However, the p-term constraint for (k, m, p)-PLA based CPLD mapping is not monotone. Specifically, if a cluster $\text{CST}_{\mathbf{R}}$ is not (k, m, p)*-feasible,* it does not necessarily imply that a supercluster of $\text{CST}_{\mathbf{R}}$ is not (k, m, p)*-feasible*. Because of this nonmonotone p-term constraint, the minimum mapping depth at each node of the network is no longer monotone [6]. As shown in Fig. 2 [6], CST_{v1} has five p-terms (covered by three (4,4,1)-*feasible* clusters) while its supercluster CST_{v} only has three p-terms. Consequently, the optimal mapping depth at node $v1$ is two, whereas the optimal depth at its successor v is just one. Moreover, PLA implementation requires two-level logic optimization, which is an NP-hard problem. The nonmonotone property and the complexity of two-level optimization make it very difficult to efficiently compute the optimal solutions. Therefore, we rely on heuristic algorithms to find a good solution. We have modified both FlowMap and DAG-Map labeling procedures for CPLD architectures and find that they produce comparable depth and area results. As the labeling method in DAG-Map is much simpler, we adopt a modified version of the DAG-Map labeling method.

The DAG-Map labeling procedure is based on Lawler's algorithm [12]. We extend it to consider the p-term constraint, as shown in Fig. 3. The label of a node v, label(v), represents the level (logic depth) of the node in the final mapped PLA network. Note that we have ignored the fanouts of the internal nonroot nodes that go out of CST_v in order to minimize the label of each node. These fanouts are named as *out-of-cluster* fanouts of CST_v. The most time-consuming part in labeling is the p-term calculation required in the $(k, m, 1)$*-feasibility* check. This

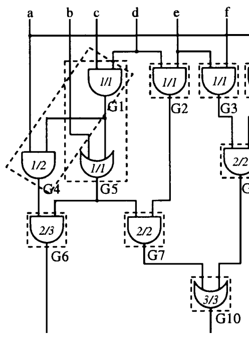

Fig. 4. Boolean network after labeling for (3,3,2)-PLAs.

is done by the two-level logic minimizer ESPRESSO [14], which generally runs very fast if k and m are small although the worst-case runtime is exponential. The number of feasibility checks is proportional to the number of circuit nodes.

After the labeling stage, the logic depth of the final PLA network has been determined. The label of each node in a cluster represents the arrival time (AT) of the output signal of that node in the corresponding PLA under the unit PLA-delay model. For area optimization later in the mapping stage, we calculate two more delay parameters for each node: required time (RT) and slack time (ST). Assume that RT of the final mapping solution is the maximum AT in the network. From POs to PIs, we trace clusters in the network and calculate RTs of internal nodes by deducting PLA-delays on the paths. The difference between RT and AT of a node is ST. A network example after the labeling stage is shown in Fig. 4, which has nine single-output clusters. The target architecture is a (3,3,2)-PLA based CPLD. Each gate in Fig. 4 belongs to some clusters and has been marked with label/RT representing its label and RT.

C. Mapping Stage

The second stage of our algorithm is to generate multioutput (k, m, p)-PLAs based on the label information of each node in the network. Since the logic depth of the final network has already been computed, the goal of the mapping stage is to minimize the area without affecting the logic depth of the network. To achieve small area, we directly utilize the multioutput feature of target PLAs to reduce node duplication. Node duplication is generally not helpful for area reduction because it tends to increase the number and size of PLA clusters and make the packing optimization less efficient.

The mapping procedure is summarized in Fig. 5. Starting from POs to PIs, a mapping list M records and updates the nodes to be considered throughout the mapping process. Initially, all of the PO nodes are put into M. During the flow, nodes are retrieved from M for mapping consideration and input nodes of mapped clusters are added into M. Prior to mapping each node in M, M is sorted in a label-decreasing ST-increasing order so that nodes on critical paths (with $ST = 0$) are considered first and other nodes on noncritical paths have more opportunities to take advantage of ST relaxation. For the network in Fig. 4, the mapping sequence is: $G10, G7, G9, G6, G2, G5, G3, G8, G4$. Similar to the labeling stage, the most time-consuming part in mapping is the (k, m, p)*-feasibility* check and the number of feasibility checks is proportional to the number of circuit nodes.

When a node v in the network is contained in a mapped cluster (PLA), we say that it is *covered* by this PLA. When a node is not contained in any PLA, it is *uncovered*. For each node v_M in list M, it can be either an uncovered node or a covered node. A covered node v_M

```
/* Stage 2: Mapping the network */
$M$ = list of PO nodes;
**while** $M$ is not empty **do**
    sort $M$ in a label-decreasing ST-increasing order;
    remove the first node $v_M$ from $M$;
    **if** $v_M$ is an uncovered node **then**
        form cluster $CST_{v_M}$ and $RCST_{v_M}$ of $label(v_M)$;
        **if** shared-node cluster merge succeeds **then**
            $CST_{\mathbf{R}}$ = the merged cluster;
        **else if** slack-time optimization succeeds **then**
            $CST_{\mathbf{R}}$ = $RCST_{v_M}$;
        **else**
            duplicate any shared nodes for $CST_{v_M}$;
            $CST_{\mathbf{R}}$ = $CST_{v_M}$;
        **end if**;
        put nodes in $input(CST_{\mathbf{R}})$ into $M$;
        sibling-merge (optional);
    **else**
        $CST_{\mathbf{R}}$ = the cluster containing $v_M$;
        **if** adding new output to $CST_{\mathbf{R}}$ succeeds **then**
            increase the number of outputs of $CST_{\mathbf{R}}$ by 1;
        **else**
            duplicate subcluster with root $v_M$ to be a new cluster;
        **end if**;
    **end if**;
**end while**;
```

Fig. 5. Mapping procedure of PLAmap.

in M implies that v_M is required to be an input of some PLA but is currently not a root node of the PLA that covers it. These two cases are described below in details. In this mapping process, PLAs with multiple outputs are formed directly by either cluster merging or root set enlarging, which represents a unique feature of our algorithm.

Case 1: v_M is an Uncovered Node: If v_M is an uncovered node with label l_{v_M}, a single-output cluster CST_{v_M} is formed to include v_M and all its predecessors with label l_{v_M}. Thus, any nodes in $\mathrm{node}(\mathrm{CST}_{v_M})$ have the label l_{v_M} while any nodes in $\mathrm{input}(\mathrm{CST}_{v_M})$ have smaller labels than l_{v_M}. From the labeling step, it is evident that CST_{v_M} is a $(k, m, 1)$*-feasible* cluster. It is possible that some nodes in CST_{v_M} have already been covered when other PLAs were formed earlier during the mapping process. In Fig. 4, when we are mapping $G4$, CST_{G4} is formed to cover both $G4$ and $G1$. However, $G1$ has already been covered by CST_{G5}. Three approaches are considered for the mapping of clusters like CST_{G4}.

a) Shared-Node Cluster Merge: Since CST_{G4} and CST_{G5} share one node, it is highly possible that they can also share some common p-terms if they are merged together. Thus, we first merge these two clusters into one multioutput cluster $\mathrm{CST}_{\{G4,G5\}}$ and check if the merged cluster is still (k, m, p)*-feasible*. If yes, CST_{G4} will no longer exist and CST_{G5} will be replaced by $\mathrm{CST}_{\{G4,G5\}}$. In this example, CST_{G4} and CST_{G5} cannot be merged because inputs of the merged cluster would exceed k ($k = 3$).

b) ST Relaxation: If approach a) fails, ST relaxation is the next step to apply. The idea of ST relaxation has been demonstrated as a good area-reduction technique in LUT-based FPGA mapping [4]. In our case, the ST relaxation is much more complicated because we have to consider p-term constraints. This step attempts to form a reduced cluster RCST_{G4} as a separate new PLA. RCST_{G4} is the subcluster of CST_{G4} that excludes any shared nodes with other clusters. In our case, RCST_{G4} contains only one node, $G4$. As RCST_{G4} is smaller than CST_{G4}, it may be further packed with other clusters later on. Several strict criteria are verified here. First, the $(k, m, 1)$-feasibility of RCST_{G4} itself needs to be checked because of the nonmonotone constraints. Second, the possibility of introducing an additional output $G1$ from CST_{G5} to become an input of RCST_{G4} is checked to ensure $\mathrm{CST}_{\{G1,G5\}}$ is still feasible. Third, this new output introduction will

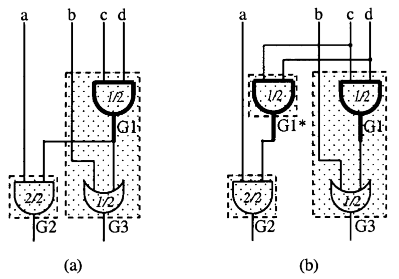

Fig. 6. Mapping case 2: (a) $G1$ is a covered node; (b) $G1$ is duplicated.

change labels/STs, and only the situation without ST violation is allowed. In our example, RCST_{G4} can be formed as a small PLA since $G4$ has a slack-time of one and CST_{G5} can provide an extra output $G1$.

c) Node Duplication: If approach b) fails, we have to duplicate those shared nodes of CST_{G4} and CST_{G5}, i.e., $G1$. A new PLA for CST_{G4} is created including $G4$ and the duplicated node $G1*$. CST_{G5} can be treated as intact except for some fanout updates. This last approach represents the worst case.

After the above mapping operation, node v_M and its originally uncovered predecessors with label(v_M) will either be covered by a newly formed cluster rooted at v_M as in approaches b) and c), or covered by a merged cluster as in approach a). At this point, an optional operation called *sibling-merge* can be applied. Sibling-merge attempts to merge the newly formed cluster with another mapped cluster with the same label. The cluster sharing the maximum number of inputs with the newly formed cluster has a higher priority for merging. Sibling-merge does not always generate a better overall result because it is a local optimization step.

Case 2: v_M is a Covered Node: If v_M is a covered node, it has to be a PLA output (root node), but is currently not identified yet. As the target is a multioutput PLA, we first check the possibility of introducing v_M as a new output of the existing PLA. An example is shown in Fig. 6 (a). $G1$ is in input(CST_{G2}). When CST_{G2} is mapped, the non-PI input $G1$ is put into the mapping list M. Later, when CST_{G3} is mapped, node $G1$ is covered by CST_{G3}. When the mapping process retrieves $G1$ from M, we try to introduce $G1$ as a new output of CST_{G3} as long as the resulting $\mathrm{CST}_{\{G3,G1\}}$ is still (k,m,p)-feasible. If it is feasible, the existing PLA is adjusted by enlarging the root set from $\{G3\}$ to $\{G3, G1\}$. Unlike the label-increasing situation in the above uncovered case, label updating is unnecessary here because the label of $G1$ is surely smaller than the label of CST_{G2}. However, if $\mathrm{CST}_{\{G3,G1\}}$ is not (k,m,p)*-feasible*, a subcluster rooted at $G1$ needs to be duplicated and becomes a new PLA with the duplicated nodes. This is the worst situation in Case 2. A duplication example is shown in Fig. 6 (b).

D. PLA Packing Stage

After the mapping stage, a network of PLA clusters has been generated. To further reduce the area without logic depth increase, two packing operations are developed.

The first operation is PLA collapsing, similar to *greedy-pack* operation in DAG-Map and the partial collapsing concept in TEMPLA. Any PLA that can be collapsed into all of its fanout PLAs (different outputs of the PLA may go into different successive PLAs) can be eliminated, provided that all PLAs remain feasible after the collapsing. This introduces another optimization problem since collapsing some PLAs into their fanout PLAs may preclude the possibility of collapsing other PLAs into their fanout PLAs. Based on the empirical results

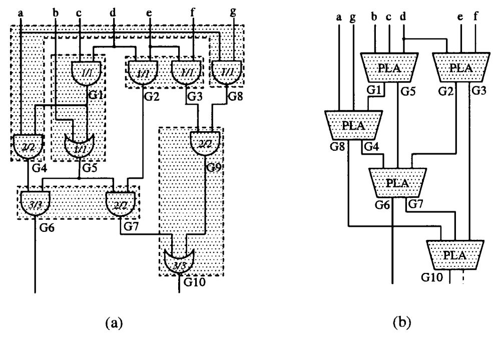

Fig. 7. Mapping solution: (a) cluster view; (b) PLA view.

in TEMPLA, our collapsing operation prefers to collapse smaller PLAs. The *size* of a PLA is defined as the product of the number of inputs $\mathrm{num\_in}$ and the number of p-terms, $\mathrm{num\_pterm}$ (i.e., $\mathrm{num\_in} * \mathrm{num\_pterm}$). PLA collapsing may decrease the logic levels of some PLAs as two serial PLAs are merged to be one PLA.

The second operation is maximum shared-input bin packing. For each PLA, a list of buckets is built based on the number of shared inputs with other PLAs. Bucket $m (m \geq 0)$ contains all PLA clusters sharing m inputs with the current PLA. In each bucket, PLAs are sorted in a size-descending order. The bucket list is traversed from the maximum shared-input bucket to the minimum shared-input bucket. The general observation is that the larger the input number shared by two PLAs, the higher the possibility they could be packed. For PLAs in each bucket, we start from the largest PLA in order to get a high packing capacity. As bin packing does not consider the logic levels of PLAs, it is possible that two PLAs with different node labels may be packed together, which is different from sibling-merge. Such a packed PLA is still reasonable because PLA is a multioutput structure with separable input-to-output paths.

The final mapping solution for our example is a PLA network consisting of only five clusters, as shown in Fig. 7. The merging of CST_{G2} with CST_{G3} is accomplished by a sibling-merge. CST_{G9} and CST_{G10} are integrated by the PLA-collapsing operation. CST_{G6} and CST_{G7}, RCST_{G4}, and CST_{G8} are packed by maximum shared-input bin packing. After packing, these two PLAs have nodes with different labels.

E. Area/Delay Tradeoff

When we generate a $(k,m,1)$*-feasible* cluster CST_v in the labeling stage, we have ignored the *out-of-cluster* fanouts of internal nodes in order to minimize the label of each node. In Fig. 4, the out-of-cluster fanout to $G4$ of the internal node $G1$ is ignored when we label and cluster CST_{G5}. It is possible that there will be many internal nodes with out-of-cluster fanouts as a cluster becomes larger and larger. A large number of those fanouts would lead to many node duplications later in the mapping stage due to the PLA output constraint. The side effect would be a larger mapping area although there is no increase in the mapping depth. To handle this problem, we have developed a threshold control procedure to trade some depth for area reduction.

For area/delay tradeoff targeting small PLA-based CPLDs, we apply a threshold control value H on the number of allowable out-of-cluster fanouts during the labeling stage. For a cluster CST_v, denote the set of the nodes with out-of-cluster fanouts as $\mathbf{F}$. If the size of $\mathbf{F}$ exceeds the threshold value H, node v will be labeled as $l+1$ even if CST_v is still $(k,m,1)$*-feasible* and v could have been labeled as l (refer to the labeling procedure in Fig. 3). H is set to be $p * \alpha$, where p is the

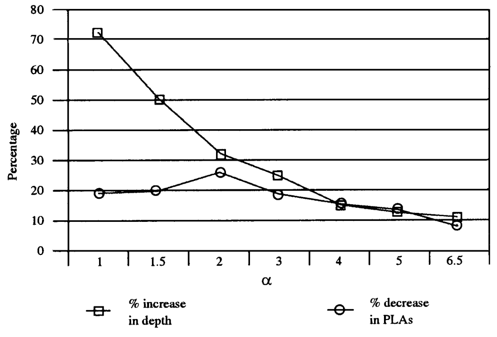

Fig. 8. Effect of threshold control parameters for $(k, m, 4)$-PLAs.

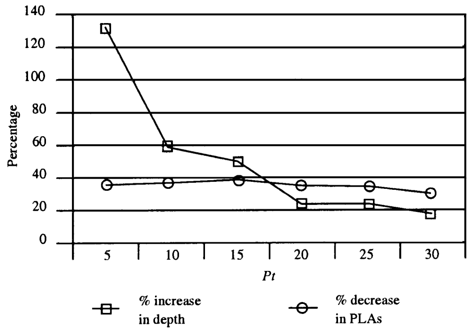

Fig. 9. Effect of different threshold values for general (36,80,16)-PLAs.

output number of (k, m, p)-PLA, and α is a user-controllable parameter. The smaller α is, the smaller H will be, and the tighter the restrictions applied to the cluster formulation. Experimental results on the effects of different values of α are shown in Fig. 8. We can see that delay increases consistently when α decreases. However, the area-reduction curve has a peak value, which indicates that overly suppressing the out-of-cluster fanouts (ends up with many small initial clusters) or overly relaxing the control (ends up with many out-of-cluster fanouts) will provide less area benefits. Instead of choosing the point with maximum area reduction, we have used $H = 12$ ($\alpha = 3$) as the default threshold control value for $(k, m, 4)$-PLAs because it offers significant area reduction without losing too much performance.

When the target is CPLDs based on large PLAs such as (36,80,16)-PLAs, our experiments reveal that the number of total p-terms plays a more crucial role. Intuitively, the number of p-terms grows faster than the number of out-of-cluster fanouts (an n-input single-output Boolean function could have as many as $3^n/n$ prime implicants [14]). In this case, the number of allowable p-terms P_t is used as the threshold control value. When the number of p-terms in CST_v exceeds P_t, node v will be labeled as $l+1$ even if CST_v is still $(k, m, 1)$-*feasible*. We carried out some empirical studies with general (36,80,16)-PLAs without any structural constraints.[1] The results are shown in Fig. 9. The area-reduction curve is flatter than that in Fig. 8, but it also has a peak value. To maintain good performance, $P_t = 20$ is chosen in this case. Since clusters are rather small after labeling compared to the capacity of the (36,80,16)-PLAs, the sibling-merge step is quite effective and the maximum shared-input bin packing also offers good area reduction.

F. Applications to Commercial CPLDs

Currently, there are several major CPLD families on the market. Altera's high-speed, high-density MAX families are based on MAX architecture [16]. Lattice's MACH 5 CPLD architecture consists of PAL blocks that allow the implementation of large equations (up to 32 p-terms) with only one pass through the logic array [18]. XILINX recently released CoolRunner-II CPLDs claiming to offer both high performance and low power [17].

In this work, we examine one type of CPLDs, Altera's MAX 7000B, which is the most widely used among MAX families. The EEPROM-based MAX 7000B family provides 6000–10 000 usable gates, 36–212 I/O pins, and up to 32 logic array blocks (LABs). Multiple LABs are linked together via the programmable interconnect array (PIA). This global bus structure provides programmable paths that could connect any signal source to any destination throughout

[1]Structural constraints for commercial CPLDs are ignored. This is to make our empirical study as general as possible. It turns out the results also apply to the Altera Multiple Array MatriX (MAX) 7000B CPLD that we are targeting (refer to Section III-F).

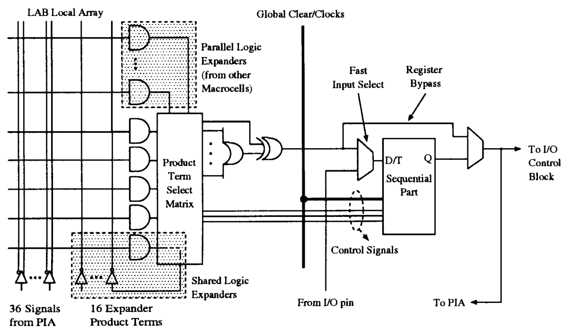

Fig. 10. MAX 7000B macrocell.

```
/* feasibility check for MAX 7000B */
$CST_{\mathbf{R}}$ = a cluster requiring feasibility check;
$P$ = integer array of length $|\mathbf{R}|$;
for each output $i$ of $CST_{\mathbf{R}}$ do
    $Pterm$ = number of p-terms at output $i$;
    if $Pterm > P_t$ then
        infeasible macrocell at output $i$;
        reject $CST_{\mathbf{R}}$;
    else $P[i] = Pterm$;
    end if;
end for;
$a = 16$;
sort elements of $P$ in descending order;
for each $P[i]$ do
    $N = P[i]/5$; /*round down*/
    if $(P[i]\ mod\ 5) > 0$ then
        /* expanders borrowed from neighbors */
        extra_expanders = $N * 5$;
        $a = a - N - 1$;
    else if $(P[i]\ mod\ 5) = 0$ then
        extra_expanders = $(N - 1) * 5$;
        $a = a - N$;
    end if;
end for;
if $a < 0$ then
    reject $CST_{\mathbf{R}}$;
else $CST_{\mathbf{R}}$ is a feasible LAB;
end if;
```

Fig. 11. Feasibility check procedure for Altera MAX 7000B.

TABLE I
EFFECT OF DIFFERENT AREA-REDUCTION TECHNIQUES

Schemes	Area Increase %
No shared-node cluster merge	0.4
No ST relaxation	5.2
No sibling-merge	7.9
No PLA collapsing	0.6
No bin packing	13.1

TABLE II
AREA/DEPTH COMPARISON OF PLAMAP AND TEMPLA

	(10,12,4)-PLAs				(12,12,4)-PLAs			
	PLAmap		TEMPLA		PLAmap		TEMPLA	
benchmarks	area/depth	runtime(s)	area/depth	runtime	area/depth	runtime	area/depth	runtime
alu4	179/4	92.9	203/8	4787.5	102/4	90.7	166/8	30584.0
dalu	108/3	35.2	65/9	205.4	86/6	45.3	54/10	585.7
ex5p	67/2	673.9	193/11	170.6	64/2	680.3	171/9	473.5
misex3	336/4	322.2	286/8	2860.0	192/3	275.9	229/8	16690.2
C5315	161/5	75.0	94/11	56.0	152/5	146.1	88/11	64.0
C7552	169/7	125.8	131/11	196.8	170/7	205.7	130/11	674.6
des	316/5	227.8	248/8	294.5	258/4	406.4	199/7	656.5
i10	208/8	85.2	165/20	87.8	196/8	114.2	141/18	136.0
i8	103/4	62.2	94/5	47.5	90/4	312.0	82/5	60.8
pair	130/4	45.8	102/9	91.5	117/4	58.3	89/8	180.2
cordic*	11/3	4.8	9/4	46.1	11/3	6.3	7/5	242.6
e64*	68/3	16.3	55/5	4.4	60/2	13.3	48/4	3.8
pdc*	504/7	527.2	501/12	1390.4	415/6	580.6	401/11	5773.9
spla*	554/5	664.63	478/12	1556.2	446/6	691.9	399/12	7103.3
table3*	114/5	40.5	82/9	47.2	106/5	169.8	67/8	132.8
Total	3028/69	2999.43	2706/142	11841.9	2465/69	3796.8	2271/135	63361.9
Comparison	1/1	1	-10.6%/+105.8%	+294.8%	1/1	1	-7.9%/+95.7%	+1568.8%

the entire device. PIA makes a design's timing performance easy to predict. Each LAB contains a group of 16 macrocells. Fig. 10 shows the structure of the macrocell. Each LAB is fed by 36 input signals from PIA. All of these signals are available within the LAB in their true and inverted form.

As shown in Fig. 10, each macrocell can be supplemented with both shareable expander p-terms and high-speed parallel expander p-terms to provide up to 32 p-terms per macrocell. Shareable expanders can be viewed as a pool of uncommitted single p-terms (one from each macrocell in the LAB) that feed back into the LAB logic array and can be shared by any or all macrocells in the LAB. Parallel expanders are unused p-terms that can be allocated to a neighboring macrocell to implement faster complex functions. Parallel expanders allow up to 20 p-terms to directly feed a macrocell OR logic, among which five default p-terms are provided by the macrocell and three sets of up to five parallel expanders per set are provided by neighboring macrocells in the LAB. The lending and borrowing of parallel expanders have some architectural constraints. Both shareable and parallel expanders incur extra delay. The whole LAB can be treated as a special (36,80,16)-PLA with structural constraints.

Here we briefly explain the changes to PLAmap with regard to the specific Altera LAB structure.

Labeling Stage. As mentioned in Section III-E, the number of p-terms P_t for PLA output is used as an effective way to achieve area/delay tradeoff for large PLAs. Just as the empirical results of the general (36,80,16)-PLAs, we find that $P_t = 20$ also produces the best results when we are targeting Altera's CPLDs. By setting $P_t = 20$, we also limit the amount of shareable expanders used since all twenty product terms can be realized without the involvement of shareable expanders. As a result, the mapping solution is further directed toward faster speed.[2]

Mapping and Packing Stage. The feasibility checks in the basic mapping and packing procedures are adapted for Altera's LAB structure. When P_t is less than or equal to 20, the feasibility check procedure is outlined in Fig. 11. When P_t is greater than 20, the procedure is slightly more complicated to account for both parallel expanders and shareable expanders borrowed from neighboring macrocells.

[2]Parallel expanders incur much less delay than shareable expanders [16].

TABLE III
DIFFERENT SYNTHESIS OPTIONS TO GENERATE MPII'S OWN RESULTS

Options	Global Syn	Multi-level Syn	Parallel Exps
noMLS_Fast	Fast	No	Yes
noMLS_Normal	Normal	No	No
MLS_Fast	Fast	Yes	Yes
MLS_Normal	Normal	Yes	No
MLS_Normal_Pexp	Normal	Yes	Yes

Macrocell Packing. Macrocell level packing is used to pack the combinational logic at the data input of a flipflop and the flipflop itself into the same macrocell. Without this step, the combinational logic and the flipflop would each occupy one macrocell in MAX 7000B and result in larger macrocell usage and longer register to register delay.

IV. EXPERIMENTAL RESULTS

A. Experimental Settings

Our program has been implemented in C language within the SIS [15] framework allowing access to existing network manipulation procedures and the ESPRESSO minimizer. In the following, we first conduct experiments to show the effects of different area-reduction techniques. We then compare PLAmap with TEMPLA for (10,12,4)-PLAs and (12,12,4)-PLAs. Next, we compare the mapping solutions of PLAmap with several sets of results generated by Altera's MAX + PLUS II using different synthesis settings.

B. Effects of Different Area-Reduction Techniques

Among the three stages of PLAmap, the labeling stage determines the network depth. The other two stages attempt to reduce the area without changing the overall depth. To understand the efficiencies of the area-reduction techniques in our algorithm, we studied the effect of each specific technique on the (10,12,4)-PLA structure with a set of Microelectronics Center of North Carolina (MCNC) benchmarks. The results are given in Table I. Each scheme with a specific technique excluded is compared to the scheme with all the techniques enabled. It can be seen that sibling-merge and bin packing are the most efficient

TABLE IV
AREA/DELAY COMPARISON OF PLAMAP AND MPII MLS_Normal_Pexp

	PLAmap				MPII-MLS_Normal_Pexp				Instance Cmp.	
Benchmarks	LAB Num	I2O delay	R2R delay	MAX (ns)	LAB Num	I2O delay	R2R delay	MAX (ns)	L A B	Delay
C1355	10	17.1		17.1	8	21.5		21.5	-20.0%	25.7%
C3540	23	34.4		34.4	21	56		56	-8.7%	62.8%
C880	12	17.6		17.6	7	27.6		27.6	-41.7%	56.8%
alu4	16	23.4		23.4	11	47.2		47.2	-31.3%	101.7%
apex4	25	13.8		13.8	16	64.6		64.6	-36.0%	368.1%
duke2	6	11.3		11.3	5	15.1		15.1	-16.7%	33.6%
ex5p	12	12.8		12.8	13	17.7		17.7	8.3%	38.3%
k2	27	13.6		13.6	26	36.5		36.5	-3.7%	168.4%
t481	8	16.7		16.7	2	19.9		19.9	-75.0%	19.2%
table5	10	12.3		12.3	7	22.5		22.5	-30.0%	82.9%
vda	10	12.3		12.3	8	20.6		20.6	-20.0%	67.5%
x4	14	8.1		8.1	13	11.3		11.3	-7.1%	39.5%
minmax10	16	34.3	21.9	34.3	8	77.9	24.6	77.9	-50.0%	127.1%
planet	8	14.4	12.7	14.4	6	21.9	15.7	21.9	-25.0%	52.1%
s1196	8	14.1	12.4	14.1	8	18.1	16.6	18.1	0.0%	28.4%
s1238	9	14.5	12.8	14.5	6	22.7	16.6	22.7	-33.3%	56.6%
s1423	23	20.7	19	20.7	9	29.3	27.2	29.3	-60.9%	41.5%
s1488	5	13.5	12.4	13.5	5	13.1	12.5	13.1	0.0%	-3.0%
s1494	5	9.8	12.4	12.4	5	13.1	12.5	13.1	0.0%	5.6%
s838	5	14.3	11.7	14.3	10	57.4	7.5	57.4	100.0%	301.4%
s9234	13	13.5	17.5	17.5	31	23.5	28.5	28.5	138.5%	62.9%
sbc	14	14.5	12.8	14.5	17	19.4	19.6	19.6	21.4%	35.2%
scf	19	15	13.6	15	17	22.9	53	53	-10.5%	253.3%
tbk	9	18.3	16.9	18.3	5	22.9	55.5	55.5	-44.4%	203.3%
Average									-10.3%	+92.9%

techniques. Both techniques give higher priorities to clusters sharing the maximum number of inputs. Sibling-merge is a localized step considering clusters with the same label only. ST relaxation also achieves good area reduction because it generates smaller clusters that provide more flexibility for latter operations. Shared-node cluster merge and PLA collapsing provide only marginal benefits. This study indicates that smaller clusters at initial stages may lead to more area reduction in later stages. Meanwhile, input sharing is the most important factor in generating area-efficient PLA blocks.

C. Comparison With TEMPLA

TEMPLA is also built in the SIS framework. The published results of TEMPLA were based on eight_bounded circuits. For an accurate comparison, we ran both PLAmap and TEMPLA on two_bounded circuits. We decomposed some of TEMPLA's published circuits into two_bounded ones and ran TEMPLA with them. The results of mapping area were actually 8.5% better compared with the original published results of TEMPLA, giving TEMPLA an advantage in this experimental setting over its original setting.

A (10,12,4)-PLA structure was used in TEMPLA and also tested in our experiment. In addition, a (12,12,4) structure is used to study the impact of PLA input numbers since the number of PLA inputs shows a major influence on the mapping results for smaller PLAs.

Fifteen benchmarks are shown in Table II. All jobs are run on a SUN Ultra 10 machine. Circuits with "*" are the original circuits used by TEMPLA that are decomposed into two_bounded networks. Area is the number of PLAs and depth is determined based on the *unit PLA-delay model*. The comparison shows that TEMPLA produces 8% to 11% less area but with twice the mapping depth as PLAmap (PLAmap reduces 50% on mapping depth). TEMPLA also has a much longer runtime, especially in the case of (12,12,4) structure.

D. Comparison With MAX + PLUS II

We compared PLAmap with MAX + PLUS II version 9.6 (MPII).[3] All results are tested on the largest MAX 7000B device, EPM7512BFC256–6, which has 32 LABs and a total of 512 macrocells. Each original logic network is first optimized by SIS and decomposed into a *two-bounded* network by *dmig* [2].

The results of PLAmap are obtained as follows. After optimization and decomposition, circuits are run through PLAmap to generate mapping solutions. In the mapped network, each node is specified either as an AND-OR cell (the combinational part in a PLA) or as a D-FlipFlop (DFF). A macrocell in MAX 7000B devices can be configured as an AND-OR combinational cell alone, an AND-OR cell with output registered by a DFF, or just a DFF itself. Macrocells are grouped together using a CLIQUE block, which will be treated by MPII as a single unit to be fit into the same LAB, if possible. The logic equation of each AND-OR cell and the specifications of DFFs are written into a *text design file (tdf)* file. The CLIQUE information is specified into an *assignment and configuration file (acf)* file. The *tdf* description is fed into MPII in a WYSIWYG style. This style directs MPII's logic synthesizer to change the logic of the circuit as little as possible during compilation by turning off many of the logic synthesis options, thereby preserving our mapping information. Thus, MPII is simply used as a fitting tool to get delay and area information for PLAmap's mapping. The results of MPII's mapping are obtained by running the unmapped *two-bounded* circuits through MPII's own synthesis and mapping procedures followed by its fitting procedure. Different synthesis options are used to explore the best performance for MPII's own results.

[3]Since the submission of this work, Altera released MPII v10.2 in the Summer of 2002, which licensed the PALACE physical synthesis tool from Aplus Design Technologies, Inc. (www.aplus-dt.com). As a result, the performance of v10.2 is significantly improved.

For combinational circuits, the delay is obtained by using MPII's timing analyzer for the longest path between primary inputs and primary outputs, denoted as I2O delay. For sequential circuits, there is another type of delay—the maximum clock period reported from MPII's registered performance window, denoted as R2R delay. We use the maximum of I2O and R2R delay as the largest circuit delay.[4] The time unit is nanosecond. Because each LAB can be treated as a special (36,80,16)-PLA, we use the number of LABs occupied in the device as the area of the design after fitting.

For MPII, we tried five combinations of different options to get MPII's best results (Table III). These five styles are the most commonly used options among MPII's customers.[5] "Fast" global synthesis style directs MPII's logic synthesizer to optimize the design for fast speed rather than minimum area. "Normal" style tries to optimize the design with minimum area without sacrificing speed. Multilevel synthesis (MLS) handles complex logic to reduce area and achieve a fit, but may produce inferior speed. Turning on parallel expanders can make each logic cell larger, potentially saving the overall mapping depth. However, it also has the potential to increase the area, making the fitting task more difficult.

MPII's own results show a dramatic difference between turning MLS on and off. With MLS on, every circuit fits into the designated device but the delay is much larger than without MLS. On the other hand, while turning off MLS improves delay in MPII, not every circuit can fit into the device due to either more complex logic or a larger area. Among the options with MLS on, $\mathrm{MLS\_Normal\_Pexp}$ gives the best delay. The detailed comparison results are shown in Table IV. The first 12 circuits are combinational and the second 12 are sequential. For each circuit type, we choose half circuits with gate levels less than 15 and the other half with levels no less than 15. Compared to PLAmap, MPII generates 10.3% less area in terms of the number of LABs and 92.9% more delay (in other words, PLAmap reduces circuit delay by 48.2%). Among the options without MLS, $\mathrm{noMLS\_Fast}$ generates the best delay for MPII. MPII generates 7.9% less area with 19.7% more delay than PLAmap. However, seven out of the 24 circuits could no longer fit into the device. The results of the other three MPII options are inferior compared to the previous two options. Individually, $\mathrm{MLS\_Fast}$ generates 8.7% less area and 107.7% more delay; $\mathrm{MLS\_Normal}$ generates 4.5% more area and 156.3% more delay; and $\mathrm{noMLS\_Normal}$ generates 16.5% more area and 42.7% more delay with six unfit circuits. It is worthwhile to mention that MPII works on reducing the number of macrocells to minimize area. Usually, the less macrocells used, the less LABs occupied by these macrocells. However, the allocated macrocells may not be tightly packed into LABs. In our experiments, there are several circuits (e.g., s838 and s9234) belonging to this scenario.[6]

V. Conclusion

We have presented a new performance-driven mapping algorithm, PLAmap, for CPLD architectures. Our algorithm breaks the technology-mapping process into three stages: labeling, mapping, and packing. The primary goal is to minimize the delay of mapped circuits. Meanwhile, we have successfully reduced the area by applying several techniques including threshold control, slack-time relaxation, and PLA packing. For CPLD architectures based on small PLAs such as (10,12,4)-PLAs, we compared our results with a previous algorithm TEMPLA. The comparison shows that TEMPLA produces 8%–11% less area but 96%–106% more depth than PLAmap, or PLAmap reduces the mapping depth by about 50%. PLAmap also has much less runtime than TEMPLA. For commercial CPLDs based on large PLAs such as (36,80,16)-PLAs with special structural constraints, we modified our program to take into account these constraints in Altera's MAX 7000B CPLD architecture. Experimental results show that Altera's MAX + PLUS II produces 10% less area but 93% more delay, or PLAmap reduces the delay by 48%.

[4]There are other types of paths such as I2R (primary input to register) and R2O (register to primary output). These two types of paths are not counted in the clock-period calculation in MPII.

[5]Information provided by Altera Corporation.

[6]Our mapping and packing procedures directly work on reducing the total number of multioutput PLAs. This, in fact, is a unique contribution of this work. If we intend to reduce the number of macrocells instead, different algorithms may be used (one example is k_m_flow[6]). When the number of macrocells is used as the area metric, MPII generates 17% to 29% less area than PLAmap based on different option combinations.

References

[1] J. H. Anderson and S. D. Brown, "Technology mapping for large complex PLD's," in *Proc. 35th ACM/IEEE Design Automation Conf.*, 1998, pp. 698–703.

[2] K. C. Chen, J. Cong, Y. Ding, A. Kahng, and P. Trajmar, "DAG-Map: Graph-based FPGA technology mapping for delay optimization," *IEEE Design Test Comput.*, pp. 7–20, Sept. 1992.

[3] J. Cong and Y. Ding, "FlowMap: An optimal technology mapping algorithm for delay optimization in lookup-table based FPGA designs," *IEEE Trans. Computer-Aided Design*, vol. 13, pp. 1–12, Jan. 1994.

[4] ——, "On area/depth trade-off in LUT-based FPGA technology mapping," *IEEE Trans. VLSI Syst.*, vol. 2, pp. 137–148, June 1994.

[5] J. Cong and Y. Huang, "Structural gate decomposition for depth-optimal technology in LUT-based FPGA designs," *ACM Trans. Design Automation Electron. Syst.*, vol. 5, no. 2, pp. 193–225, 2000.

[6] J. Cong, H. Huang, and X. Yuan, "Technology mapping for k/m-macrocell based FPGA's," in *Proc. ACM/SIGDA Int. Symp. Field Programmable Gate Arrays*, San Jose, CA, Feb 2000, pp. 51–59.

[7] R. J. Francis, J. Rose, and Z. Vranesic, "Chortle-crf: Fast technology mapping for lookup table-based FPGA's," in *Proc. 28th ACM/IEEE Design Automation Conf.*, 1991, pp. 227–233.

[8] Z. Hasan, D. Harrison, and M. Ciesielski, "A fast partition method for PLA-based FPGA's," *IEEE Design Test Comput.*, pp. 34–39, Dec. 1992.

[9] D. Kania, "A technology mapping algorithm for PAL-based devices using multi-output function graphs," in *Proc. 26th Euromicro Conf.*, Sept. 2000, pp. 146–153.

[10] J. Kim, H. Kim, and C. Lin, "A new technology mapping for CPLD under the time constraint," in *Proc. ASP-DAC*, Feb. 2001, pp. 235–238.

[11] J. L. Kouloheris, "Empirical Study of the Effect of Cell Granularity on FPGA Density and Performance," Ph.D. dissertation, Stanford Univ., Stanford, CA, 1993.

[12] E. L. Lawler, K. N. Levitt, and J. Turner, "Module clustering to minimize delay in digital networks," *IEEE Trans. Comput.*, vol. C18, pp. 47–57, Jan. 1969.

[13] S. Liu, M. Pedram, and A. M. Despain, "$PLATO\_P$: PLA timing optimization by partitioning," in *Proc. IEEE Symp. Circuits Syst.*, vol. 3, 1995, pp. 1744–1747.

[14] G. D. Micheli, *Synthesis and Optimization of Digital Circuits*. Toronto, ON, Canada: McGraw-Hill, 1994.

[15] E. Sentovich *et al.*, "SIS: A System for Sequential Circuit Synthesis," Univ. California, Berkeley, CA, Memo. no. UCB/ERL M92/41, 1992.

[16] *MAX 7000B Programmable Logic Device Family, the Altera Data Book*, Altera Corporation, 2000.

[17] *CoolRunner-II CPLD Family Data Book*, XILINX, 2002.

[18] *The Lattice Data Book*, Lattice Semiconductor Corporation, 2000.