Performance-Driven Mapping for CPLD Architectures

Deming Chen, Jason Cong,
Miloš D. Ercegovac, and Zhijun Huang
Department of Computer Science
University of California, Los Angeles
Los Angeles, CA 90095

{demingc, cong, milos, zjhuang}@cs.ucla.edu

ABSTRACT

In this paper we present a performance-driven mapping algorithm, PLAmap, for CPLD architectures which consist of a large number of PLA-style logic cells. The primary goal of our mapping algorithm is to minimize the depth of the mapped circuit. Meanwhile, we have successfully reduced the area of the mapped circuits by applying several heuristic techniques, including threshold control of PLA fanouts and product terms, slack-time relaxation, and PLA-packing. We compare our PLAmap with a recently-published algorithm TEMPLA [1] and a commercial tool, Altera's MAX+PLUS II [16]. Experimental results on various MCNC benchmarks show that overall TEMPLA uses 8 to 11% less area at the cost of 96 to 106% more mapping depth, and MAX+PLUS II uses 12% less area but 58% more delay compared with our mapper.

Keywords

CPLD, FPGA, PLA-style logic cells, technology mapping, delay optimization.

1. INTRODUCTION

Programmable Logic Devices (PLDs) have been widely used for the implementation of digital circuits due to their instant manufacturing turnaround, low start-up costs, and ease of design changes. There are two major types of PLDs: Field Programmable Gate Arrays (FPGAs) and Complex Programmable Logic Devices (CPLDs). Most FPGAs have logic cells based on look-up-tables (LUTs), and some have multiplexer-based or gate-based logic cells. CPLDs are based on PLA-style logic cells, which are also referred as *p-term blocks* or simply PLAs. PLAs are considered to be coarse-grained logic cells because they typically have a large number of inputs and outputs, and hence can realize a large number of different logic functions. In contrast, FPGAs use small programmable cells, usually LUT cells with 4 or 5 inputs, which can produce higher logic densities. Their success is further propelled by a great deal of algorithmic study and tool development [5]. Although CPLDs provide only medium density, they are faster than FPGAs because PLAs are much larger than LUTs, so the CPLD implementation results in fewer levels of logic. The worst-case delay incurred by CPLDs also tends to be more predictable because PLAs within a CPLD device communicate directly through crossbar-like programmable interconnection structures.

Kouloheris and El Gamal [11] investigated the best granularity for CPLDs and found that the area of the CPLD would be the smallest if each PLA had 8 to 10 inputs, 3 to 4 outputs and 12 to 13 product terms. However, as we will see in the latter part of this paper, commercially available CPLDs use much larger PLAs as their logic blocks. Large PLA blocks help CPLD devices to provide high speed and also make the interconnection easier so that predictable timing can be guaranteed. Nevertheless, large PLAs increase the difficulties of logic synthesis and technology mapping due to the NP-hard complexity of the two level minimization problem. However, as more and more logic gates are being integrated into one single chip and the drive for performance requires high chip speed and low noise, PLA-style CPLD devices show a promisingly bright future in the PLD industry. Cong *et al.* [7] showed that PLDs based on single-output PLA-like macrocells could outperform LUT-based FPGAs in terms of both delay and area. Khatri *et al.* [10] explored cross-talk immune VLSI design using a network of PLAs embedded in a regular layout fabric. In their approach, the logic netlist was implemented in the form of a network of medium-sized PLAs. Regular layout for PLA logic and routing regions between PLA blocks was designed to be highly cross-talk immune. The crosstalk immunity, high speed, low area overhead and high predictability of their methodology indicated that the PLA-network based VLSI architecture is promising in the Deep Sub-Micron (DSM) era [10]. These new trends towards CPLDs and the increasing complexity of CPLD devices call for new effective and highly automated CAD tools that achieve good performance, maximize logic utilization, and continue to produce the ease-of-design and fast time-to-market benefits [6].

In contrast to extensive studies on FPGA mapping algorithms, limited work has been targeted for CPLD mapping technologies. There is almost no solid research done from the perspective of performance optimization. Hasan *et al.* [9] proposed a fast heuristic partition method for PLA-based structures. Kouloheris presented DDMap [12] which adapted a LUT-based technology mapper and set the number of LUT inputs to the number of PLA-style block in-

FPGA 2001, February 11-13, 2001, Monterey, CA, USA.

puts. Then, any node containing more product-terms than allowable in the PLA was decomposed into smaller nodes. Finally, the nodes are packed into multi-output PLA-style blocks. Recently, 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 was similar to that of the Chortle-crf technology mapper [8] for LUT-based FPGAs. Another related work, *k_m_flow* [7], was a technology mapper for single-output PLA-like macrocells.

In this paper, we present a performance-driven mapping algorithm for CPLDs, called *PLAmap*. In the following sections, we will use p-term blocks and PLAs interchangeably. Each PLA has the structure shown in Figure 1. A (k, m, p)-PLA implies a PLA with k inputs, m product terms and p outputs. The primary goal of our mapping algorithm is to minimize the delay/depth of mapped circuits. We have also successfully reduced the mapping area by applying several heuristic techniques. For CPLD structures with a large number of small PLAs such as $(10, 12, 4)$-PLAs, we compared our approach with TEMPLA. To demonstrate that our algorithm could also be applied to commercial CPLDs, we modified our program to take into account structural constraints of the p-term block in one commercial CPLD device, Altera's MAX 7000B [16]. Experimental results on various MCNC benchmarks show that PLAmap can achieve a much better delay with just small area overhead when compared with both TEMPLA and Altera's MAX+PLUS II CAD tool.

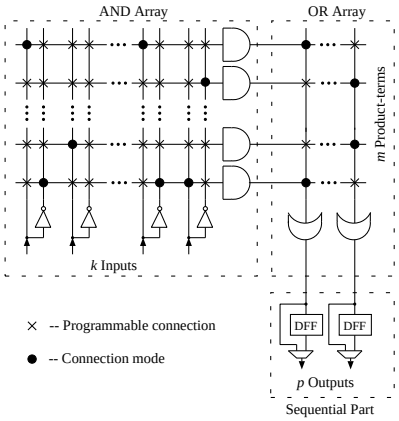


Figure 1: (k, m, p)-PLA structure

The rest of the paper is organized as follows. Section 2 defines terminology and formulates the problem. Details of the algorithm description are given in Section 3. Section 4 gives the experimental results. Conclusions and discussions of future work are given in Section 5.

2. 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 if there is a directed path from u to v, and on the other hand, node v is a *successor* of u. We use *input(v)* to denote the set of nodes that supply inputs to node v. We assume the incoming network for PLAmap is *2-bounded*, that is, for each node v in the network, $|input(v)| \leq 2$.

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

A cluster $CST_{\mathbf{R}}$ can be optimized into a PLA as shown in Figure 1. The number of product terms of the optimized PLA is defined as the number of product terms of $CST_{\mathbf{R}}$, denoted as $pterm(CST_{\mathbf{R}})$. A cluster $CST_{\mathbf{R}}$ is said to be *(k, m, p)-feasible* if and only if $|input(CST_{\mathbf{R}})| \leq k$, $pterm(CST_{\mathbf{R}}) \leq m$ and $|output(CST_{\mathbf{R}})| \leq p$ are all satisfied. Otherwise, it is *(k, m, p)-infeasible*.

The *level* of a cluster $CST_{\mathbf{R}}$, $level(CST_{\mathbf{R}})$, is the maximum number of clusters that a path from a PI to nodes in $output(CST_{\mathbf{R}})$ needs to go through. All nodes in $node(CST_{\mathbf{R}})$ have the same *level*. Two factors determine the delay of a CPLD circuit: delay in p-term blocks and delay in interconnection paths. Because layout information is not available at this mapping stage, we assume that each interconnection edge contributes a constant delay, which is reasonable in CPLD structures. If each cluster in a network is transformed into a PLA, we can then simply approximate the circuit delay 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 longest path contributes one unit PLA-delay towards the logic depth of the network.

The technology mapping problem for CPLDs is to cover a given Boolean network with *(k, m, p)-feasible* clusters, which then are converted to PLAs. Note that we do not require these clusters to be disjoint since we allow network nodes to be duplicated if necessary as long as the resulting network is logically equivalent to the original. A mapping solution S is a DAG where each node of the DAG is *(k, m, p)-feasible*, and the edge $(CST_{\mathbf{R1}}, CST_{\mathbf{R2}})$ exists if $v \in output(CST_{\mathbf{R1}})$ is in $input(CST_{\mathbf{R2}})$. Our main objective is to compute a mapping solution that minimizes circuit delay. Secondly, we use several techniques to reduce the number of PLAs needed as much as possible.

3. ALGORITHM DESCRIPTION

3.1 Overview

Our algorithm 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; 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 *2-bounded* network. Actually, as long as each node in the input network is (k, m, p)-feasible, the network can be handled by PLAmap directly. We adhere to the *2-bounded* network because we want to stick with the same starting point for every input network. In addition, smaller gates will be more easily packed for area optimization [3].

3.2 Labeling Stage

Labeling stage determines each node's *level* and provides clustering information for the subsequent mapping step. To minimize depth in the final PLA network, we label as if the target structure only consists of $(k, m, 1)$-PLAs so that we can form a PLA cluster as deep as possible. In mapping stage, we will try to introduce new outputs from these $(k, m, 1)$-PLAs to generate (k, m, p)-PLAs ($p > 1$).

Both DAG-Map [2] and FlowMap [3] used labeling techniques as their first step to generate depth information. Although FlowMap offered an elegant polynomial time algorithm to solve the depth optimization problem optimally for LUT-based FPGA mapping, it was shown that the optimal mapping depth with product-term constraint could no longer be derived from FlowMap because of the *non-monotone* property of the minimum mapping depth at each node [7]. We can also easily show that the clustering constraints for (k, m, p)-PLA based CPLD mapping are not monotone. That is, that a cluster $CST_{\mathbf{R}}$ is not (k, m, p)-*feasible* does not imply that the superclusters of $CST_{\mathbf{R}}$ are not (k, m, p)-*feasible* either. These non-monotone constraints introduce a great difficulty in computing the optimal solutions efficiently. Basically, we will develop heuristic algorithms to tackle this NP-hard problem. To take advantage of the success of FlowMap and DAG-Map for LUT-based FPGA mapping, we modified the labeling procedures defined in FlowMap and DAG-Map to serve our purpose. We found that both of them produced comparable depth and area results for our case. Because the labeling method in DAG-Map was much simpler without losing efficiency, a modified version of the DAG-Map labeling method was adopted.

The DAG-Map labeling method is based on Lawler's algorithm [13]. We have extended it to consider the product-term constraint. A label, $label(v)$, is assigned to each node v of the original network. The nodes are labeled in a topological order starting from the PIs. The topological ordering guarantees that every node is processed after all of its predecessors have been processed. Each PI node is assigned the label 0. If node v is not a PI node, let l be the maximum label of all the fanin nodes of node v. The set of node v and all its predecessors with label l form a tentative cluster CST_v. If CST_v is $(k, m, 1)$-*feasible*, the label of v is assigned as l, i.e., $label(v) = l$; Otherwise, $label(v) = l + 1$. After this process, it is evident that for each node v with label l_v, the cluster CST_v is $(k, m, 1)$-feasible. $node(CST_v)$ consists of the root v and all its predecessors with label l_v. Label l_v will be the level (logic depth) of all the nodes in CST_v in the labeled network.

When the labeling step is finished, the delay information will be updated. 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. The required time (RT) of the final mapping solution is assumed to be the maximum AT in the network, which is the logic depth of the final mapped network. Slack time (ST) of a node is defined as the difference between RT and AT. When we trace clusters in the network from POs to PIs, we can calculate RT and ST for each node. The slack-time information can be used in the mapping stage for area optimization. A network example after the labeling stage is shown in Figure 2, which has nine single-output clusters. The final target structure is $(3, 3, 2)$-PLA based CPLD. Each gate in Figure 2 belongs to some clusters and has been marked with $label/RT$ representing its label and required-time.

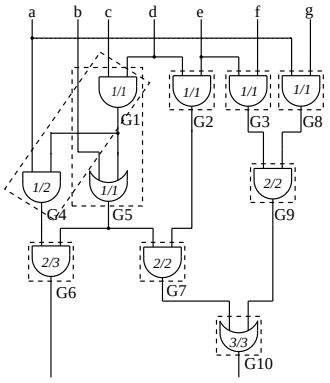


Figure 2: Boolean network after labeling

Note that when we generate a $(k, m, 1)$-*feasible* cluster CST_v, in order to minimize the label at each node, we ignore the fanouts of the internal nodes that go out of CST_v. We call these fanouts *out-of-cluster* fanouts of CST_v. In Figure 2, for example, the out-of-cluster fanout to $G4$ of internal node $G1$ is ignored when we label and cluster CST_{G5}. It is possible that as CST_v becomes larger and larger, there will be many internal nodes of $node(CST_v)$ with out-of-cluster fanouts. We can predict that a large percentage of those fanouts will incur node duplications later because of the PLA output constraint. The side effect will be a larger mapping area. We have developed a threshold control procedure to reduce this side effect, which will be covered in detail in the area/delay tradeoff section, Section 3.5.

3.3 Mapping Stage

The second stage of our algorithm is to generate (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 decided, the goal of the mapping stage is to minimize area without affecting the logic depth of the network.

The mapping process moves from POs to PIs. A mapping list M records and updates the nodes to be considered throughout the process. Initially, all of the PO nodes are put into M and more nodes (as inputs of mapped clusters) are added in along the way.

Prior to mapping each node in M starting from the beginning, M is sorted in label-decreasing, slack-time-increasing order so that nodes on critical paths (with slack time 0) tend to be considered first, and the neighborhood non-critical nodes have more opportunities to take advantage of slack time relaxation. For the network in Figure 2, the mapping

sequence is: $G10, G7, G9, G6, G2, G5, G3, G8, G4$. For each node v in M, there will be two possible situations: v is an uncovered node, or v has already been covered. We will explain these two situations in more details below.

3.3.1 Case 1: v is an uncovered node

First, if v is an uncovered node with label l_v, a single-output cluster CST_v is formed to include v and all its predecessors with label l_v. Therefore, any nodes in $node(CST_v)$ have the label l_v while any nodes in $input(CST_v)$ have the labels smaller than l_v. From the labeling step, it is evident that CST_v is a $(k, m, 1)$-*feasible* cluster. It is possible that some nodes in CST_v have already been covered when other PLAs were formed during the mapping process. In Figure 2, 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 CST_{G4} in the example.

(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 product terms if they are merged together. So we will first merge these two clusters into one multiple-output cluster CST_{G4G5} and check if the merged cluster is still (k, m, p)-*feasible*. If the answer is yes, CST_{G4} will no longer exist and CST_{G5} will be replaced by CST_{G4G5}. In this example, CST_{G4} and CST_{G5} can not be merged together because inputs of the merged cluster will exceed 3.

(B) **Slack-time (ST) relaxation**. If approach (A) fails, ST relaxation is the next approach to try. This approach 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, $RCST_{G4}$ only contains one node, $G4$. Because $RCST_{G4}$ is smaller than CST_{G4}, it may be further packed with other clusters later. However, several strict criteria need to be verified. Due to the non-monotone property of the PLA constraints, the $(k, m, 1)$ feasibility of $RCST_{G4}$ itself needs to be checked first. Furthermore, we have to determine if additional output can be brought out from CST_{G5} to provide input to $RCST_{G4}$. Since CST_{G5} and $RCST_{G4}$ originally have the same label, the above new output addition will make the label of $RCST_{G4}$ increase by 1. The label change of $RCST_{G4}$ also affects the labels and STs of its successors and the RTs and STs of its predecessors. Since we plan to keep the overall network depth unchanged in the mapping stage, the label increase of $RCST_{G4}$ is allowed only when there is no ST violation (i.e., new STs of affected nodes are still ≥ 0). Hence, delay information needs to be updated. Also, we have to ensure that no extra (k, m, p)-*infeasible* clusters are generated as a side effect. In our example, $RCST_{G4}$ can be formed as a small PLA since $G4$ has a slack-time of 1 and CST_{G5} can provide an extra output $G1$.

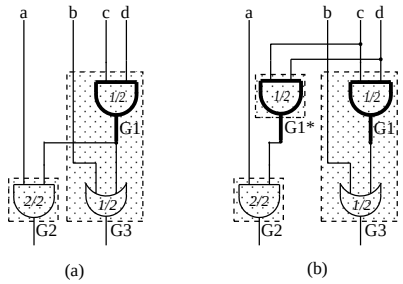
The idea of ST relaxation has been demonstrated as a good area reduction technique in LUT-based FPGA mapping [4]. In our case, the ST relaxation is much more complicated and it can produce 6 to 10% area reduction.

(C) **Node duplication**. If approach (B) fails, we then have to duplicate those shared nodes of CST_{G4} and CST_{G5}, i.e., $G1$. A new PLA for CST_{G4} is then 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 and its originally uncovered predecessors with $label(v)$ will be either covered by a newly formed cluster rooted at v 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 tries to merge the newly formed cluster with another mapped cluster of the same depth and with the maximum number of shared inputs. It can be treated as a local area optimization step that mingles with the mapping stage. Since our overall algorithm includes a global packing step after mapping, the flow with sibling-merge doesn't always generate better results compared to the flow without sibling-merge. Although sibling-merge can eliminate one PLA by merging it into another PLA, the resulting larger PLA may not have any chance to be further packed with other PLAs later.

3.3.2 Case 2: v is a covered node

Secondly, v is a covered node in some mapped PLA but is not a root node of that PLA. An example is shown in Figure 3(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}. However, $G1$ is still in the mapping list M.



(a) $G1$ is a covered internal node
(b) $G1$ is duplicated if PLA output introduction fails

Figure 3: Example of case 2

At this point, we will first try to introduce $G1$ as a new output of CST_{G3} as long as the resulting CST_{G3G1} is still (k, m, p)-feasible. Unlike the label increasing situation in the uncovered case, no label update is necessary here because the label of $G1$ is surely smaller than the label of CST_{G2}. If CST_{G3G1} is not (k, m, p)-*feasible*, another worst case is encountered: a subcluster rooted at $G1$ needs to be duplicated and becomes a new PLA with the duplicated nodes. A duplication example is shown in Figure 3(b).

3.4 PLA Packing Stage

To further reduce mapping area, two packing algorithms are developed. They also reduce the number of PLAs without depth sacrifice.

The first operation is PLA collapsing, which is 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 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, num_in and the number of product terms, num_pterm (i.e., $num\_in * num\_pterm$). Experiments show that only single-output single-fanout PLA collapsing can help in reducing area by 5 to 7%. The cost of collapsing multiple-fanout PLAs would overturn the benefits.

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 that share m inputs with the PLA. In each bucket, the PLAs are sorted in descending order according to their size, which has the same definition as in the PLA collapsing operation. Next, each bucket is traversed from the maximum shared-input bucket to the least shared-input one. And for every bucket, we try to pack the PLAs starting from the largest size down. The general observation is that the larger input number a PLA shares with another PLA, the higher the possibility they can be packed. Also, the larger size a PLA has, the better packing capacity it produces after packing with another PLA.

The final mapping solution for our example is a PLA network consisting of only five clusters in Figure 4. Figure 5 shows a structural view of the mapped results with $(3, 3, 2)$-PLAs. The merging of CST_{G6} with CST_{G7}, and CST_{G2} with CST_{G3}, is accomplished by sibling-merge. CST_{G9} and CST_{G10} are put into one cluster by the PLA-collapsing operation. $RCST_{G4}$ and CST_{G8} are packed by maximum shared-input merge. From Figure 5, we see that different outputs of one PLA can have different labels or levels. However, we can verify that the packing step will not increase the total depth of the network.[1]

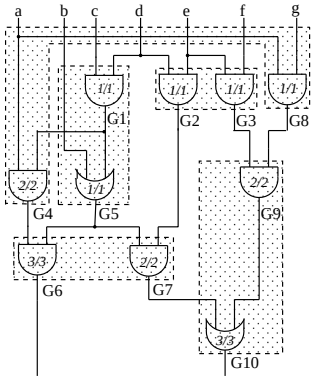


Figure 4: Network after mapping and packing

Among the three stages of PLAmap, the labeling stage determines the network depth. The other two stages target reducing area without changing overall depth. In next section, we will discuss two ways to control area/delay tradeoff.

3.5 Area/Delay Tradeoff

Area/delay tradeoff implies that some depth can be sacrificed to achieve better area. In the case of pursuing area reduction for small PLA-based structures, we used a threshold

[1] The AND-OR levels between each individual PI and PO remain the same before and after the packing operation.

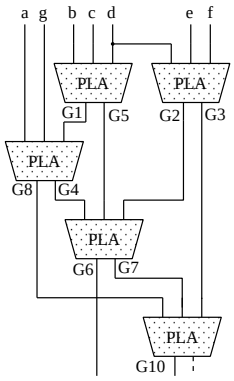


Figure 5: Final PLA network

control value ρ over the number of allowable out-of-cluster fanouts during labeling stage. Along the labeling process, when CST_v is growing larger, it starts to contain some internal nodes (other than v) with out-of-cluster fanouts. Let us denote the set of these nodes as $\mathbf{F}$. When the size of $\mathbf{F}$, $|\mathbf{F}|$, exceeds a threshold value H, we will stop the labeling process for CST_v even before it actually reaches the $(k, m, 1)$-*infeasible* point. H is calculated as $H = p/\rho$, where p is the output number of (k, m, p)-PLA. When $|\mathbf{F}| > H$, CST_v will stop growing and be counted as a single-output PLA. The larger ρ is, the smaller H will be, so the higher the restrictions applied to the cluster formulation. Experiment results on the effects of different ρ are shown in Table 1. From the table, we can see that for $(k, m, 4)$-PLAs ρ of 0.5 provides the best area reduction but with larger depth sacrifice. We have used 0.33 as our default threshold value; that is, $H = 4/0.33 = 12$. So $|\mathbf{F}|$ can not exceed 12 for this case.

ρ	Average % decrease in # of PLAs	Average % increase in depth
0.15	8.1	10.9
0.2	14.1	13.0
0.25	15.5	15.2
0.33	18.7	26.1
0.5	26.7	32.6
0.67	19.8	50.0
1.0	19.4	71.7

Table 1: Effect of different threshold control values for $(k, m, 4)$-PLAs compared to non-threshold-control case

When the circuits are large, threshold-controlled PLAs lead to significantly less duplication in the mapping step. In addition, when the PLAs are restricted to be smaller in the labeling stage, they are more capable of being packed with other PLAs in the PLA packing stage.

When the target is for CPLDs that are based on large PLAs, such as (33,80,16)-PLAs or (36,80,16)-PLAs, our experiments reveal that the number of total product terms is playing a more crucial role. Therefore, we used a different threshold parameter, number of product terms allowed for each PLA output, denoted as P_t. When the number

of product terms for output v in CST_v reaches the threshold value P_t, CST_v stops growing during the labeling stage and is set as a single-output PLA. We carried out some empirical studies with general $(36, 80, 16)$-PLAs without any structural constraints.[2] The results are shown in Table 2. We can see that when $P_t = 20$, we have the best tradeoff for area and delay in this general case. Since clusters are rather small after labeling, the sibling-merge step is quite effective and the maximum shared-input bin packing also offers good reduction for area.

P_t	Average % decrease in # of PLAs	Average % increase in depth
5	36.9	131.8
10	38.2	59.1
15	39.6	50.0
20	36.4	22.7
25	36.0	22.7
30	31.6	18.2

Table 2: Effect of different threshold values for general $(36, 80, 16)$-PLA structure compared to the non-threshold-control case

3.6 Extension to Commercial CPLDs

Several major CPLD families currently exist on the market. Altera's high-speed, high-density MAX families are based on Multiple Array MatriX (MAX) architecture [16]. Lattice's MACH 5 CPLD architecture consists of PAL blocks that allow the implementation of large equations (up to 32 product terms) with only one pass through the logic array [19]. Cypress recently released their high density Delta39K devices which is fast enough to implement a fully synthesizable 64-bit, 66-Mhz PCI core [18]. Other major CPLD vendors include XILINX and Atmel.

In this research, we will examine one type of CPLD, Altera's MAX 7000B, which is considered the best or the most sophisticated of the MAX families. The EEPROM based MAX 7000B family provides 6,000-10,000 usable gates, 36-212 I/O pins, and up to 32 Logic Array Blocks (LABs). Each LAB contains a group of 16 macrocells. Figure 6 shows the structure of the macrocell and its local array. Each LAB is fed by 36 input signals from the interconnect array. All of these signals are available within the LAB in their true and inverted form.

As shown in Figure 6, each macrocell can be supplemented with both shareable expander product terms and high-speed parallel expander product terms to provide up to 32 product terms per macrocell. Shareable expanders can be viewed as a pool of uncommitted single product 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 product terms that can be allocated to a neighboring macrocell to implement faster complex functions. Parallel expanders allow up to 20 product terms to directly feed a macrocell OR logic, with five default product terms provided by the macrocell and three sets of up to five parallel expanders per set provided by neighboring macrocells in the LAB. The lending and borrowing of parallel expanders have to follow some architecture constraints. Both shareable and parallel expanders incur an extra small delay. The whole LAB can be treated as a special $(36, 80, 16)$-PLA with structural constraints.

Multiple LABs are linked together via the Programmable Interconnect Array (PIA). This global bus is a programmable path that connects any signal source to any destination throughout the entire device. The PIA makes a design's timing performance easy to predict.

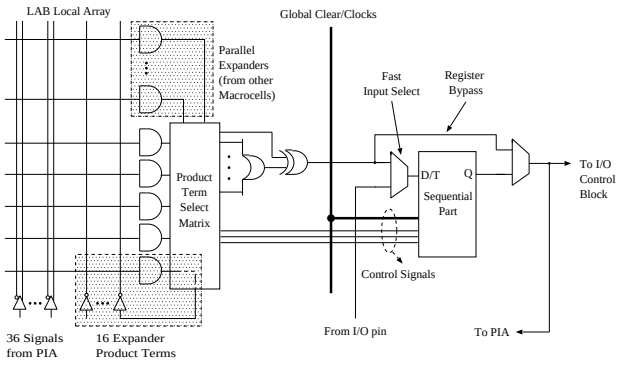


Figure 6: MAX 7000B macrocell and local array

We will briefly explain the algorithm changes to fit Altera's LAB specification.

Labeling Stage. As mentioned in Section 3.5, number of product terms P_t for PLA output is used as an effective way to achieve area/delay tradeoff for large PLAs. Using the empirical results of the general (36,80,16)-PLAs as a guideline, we find that the maximum twenty product terms per output also performs the best when we are targeting for Altera's CPLDs. By setting $P_t = 20$, we also limit the amount of shareable expanders used because all the twenty product terms can be realized without the involvement of shareable expanders, and as a result, the mapping solution is further directed towards faster speed.[3]

Mapping and Packing Stage. The mapping and packing algorithm is modified to adapt Altera's LAB structure. It is outlined as follows:

Collapse each newly formed cluster into an one-level network, in which each node represents a function of sum of product terms (p-terms). Each node belongs to the root set **R**, and is an output of the cluster.

Find the number of p-terms of each output i (a potential macrocell) and put it in an array P.

If $P[i] > P_t$, the output can not be a macrocell, so the whole cluster is rejected.

Otherwise, sort P in descending order; for each $P[i]$ starting from the beginning, calculate $N = P[i]/5$. If $(P[i]\ mod\ 5) > 0$, $N * 5$ is the number of the parallel expanders that

[2]That means, during the formulation of CST_v, the output v can use up all the eighty product terms without being concerned if it is practical or not. Besides, it does not need to worry about other structural constraints as described in Section 3.6. This case is actually used as the non-threshold-control base-case to evaluate the effects of different threshold values.

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

the corresponding output needs to borrow from its neighborhood; deduct N off the available outputs of the cluster. If $(P[i] \ mod \ 5) = 0$, deduct $N-1$ off the available outputs of the cluster.

If there are leftover p-terms, i.e., $(P[i] \ mod \ 5) > 0$, an additional constraint is applied.[4]

If the output constraint can be satisfied, the cluster can be converted into a LAB of MAX 7000B; otherwise, the cluster is rejected.

4. EXPERIMENTAL RESULTS

4.1 Experimental Settings

Our program has been implemented in C language within the SIS [15] framework so we can easily access existing network manipulation procedures and the ESPRESSO minimizer.

We first compared PLAmap with TEMPLA for $(10, 12, 4)$-PLAs and $(12, 12, 4)$-PLAs. Next, with the modified algorithm, we compared the mapping solution of PLAmap with that from Altera's MAX+PLUS II (or MPII) tool.

4.2 Comparison with TEMPLA

TEMPLA is also based on the SIS framework and relies on the I/O routines and ESPRESSO minimizer. The published results of TEMPLA were based on *8_bounded* circuits. To have a fair comparison, we decided to run both PLAmap and TEMPLA on *2_bounded* circuits. We decomposed some of TEMPLA's published circuits into *2_bounded* ones and ran TEMPLA with them. The results on mapping area were actually 8.5% better compared with the original published results of TEMPLA. So TEMPLA actually would gain an advantage in its results in this experimental setting.

A $(10, 12, 4)$-PLA structure was used in TEMPLA and hence was introduced in our experiment. Also, since the number of PLA inputs is showing a major influence upon the mapping for small PLAs, $(12, 12, 4)$ structure is also used for the experiment.

Fifteen benchmarks are shown in Table 3. All the jobs are run on a SUN Ultra 10 machine. Circuits with '*' are the original circuits used by TEMPLA and decomposed by us into *2_bounded* networks. Area is the number of PLAs and depth is decided based on the *unit PLA-delay model*. The comparison shows that TEMPLA produces 8 to 11% less area but about two times as much delay as PLAmap. Moreover, TEMPLA consumes a huge running time, especially in the case of $(12, 12, 4)$ structure.

4.3 Comparison with Altera

We carry out our experiments in version 9.6 of MPII. All results are run using the EPM7512BFC256-6 device, which has 32 LABs and a total of 512 macrocells.

The results of PLAmap are obtained as follows. First, circuits are run through PLAmap to generate mapping solutions. Then, each node of the resulting network is specified as a LCELL(logic cell), which is basically a macrocell in a MAX 7000B device. LCELLs in the same PLA are grouped by using a CLIQUE, which will be treated by MPII as a single unit to be fit into the same LAB if possible. The logic equation of each LCELL is provided in the tdf file (Text Design File), and the CLIQUE information is stored in an acf file (Assignment and Configuration File). The tdf is used as input to Altera's MPII in a WYSIWYG style (What You See Is What You Get). 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. By such, our mapping information is preserved. Area is represented by the number of LABs used, and delay is represented as the delay of the longest path.

The results of Altera are obtained by running the unmapped circuits with different synthesis options of MPII to achieve the best results for Altera. Area and delay are calculated by the same criteria as PLAmap. We have tested three sets of benchmarks:

BSet_1. 24 original MCNC benchmarks without any optimization.
BSet_2. use BSet_1 as initial benchmarks and optimize them by SIS.
BSet_3. use BSet_2 as initial benchmarks and decompose them into 2*_bounded* circuits by *dmig*.

For synthesis options, we have tried the following three different styles:

Fast. "Fast" global synthesis style, multi-level synthesis for MAX 7000, optimization for speed (index 10), full minimization, and parallel expanders.
Normal. "Normal" global synthesis style, multi-level synthesis for MAX 7000, optimization for speed (index 10), full minimization, and parallel expanders.
NoMLS. "Normal" global synthesis style, no multi-level synthesis for MAX 7000, optimization for speed (index 10), full minimization, and parallel expanders.

Among all the situations, MPII produces the best delay on average for Altera using **BSet_3** with **NoMLS** style. Turning off multi-level synthesis guarantees that MPII's synthesizer won't reduce the area by increasing the mapping level. However, as a result, 30% of the circuits can no longer fit into the device. For those unfitting circuits, we then use **BSet_3** with **Normal** style as the second best choice for Altera to fill up the rows. The comparison is shown in Table 4. The first 12 circuits are sequential and the second 12 are combinational. For each type of circuits, we choose half of them with gate level less than 15 and half of them larger or equal to 15. The circuits with '*' are unfitting circuits during the first try for Altera.

We can see that Altera generates solutions of 12% less area and 58% more delay than PLAmap. Altera performs better when the gate level is small. Among the circuits with similar levels, circuits with smaller sizes also help Altera to bring about better results. Altera outperforms PLAmap more often when dealing with sequential circuits (4 out of 12).

The *MC Level* (macrocell level) columns indicate how many steps or levels a signal goes through in the floorplan from an input pin to an output pin on the critical path. The floorplan information is available after placement and routing. We can describe each such step as one edge connecting two points along the path.

In our mapping solution, it rarely happens that an output

[4]For example, if two p-terms are loaned out, then three p-terms are left for the lending macrocell to use by itself. If that macrocell's function can be implemented using three p-terms, the macrocell will still be able to output a logic function. Otherwise, those three p-terms are wasted.

	10-12-4 PLAs				12-12-4 PLAs			
	PLAmap		TEMPLA		PLAmap		TEMPLA	
benchmarks	area/depth	runtime	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%

Table 3: Area/depth comparison of PLAmap and TEMPLA.

of a PLA is also an input of the same PLA. Also, since we are following topological order during labeling, the label of a node will never exceed the label of its successors. This guarantees that a path generally starts from the input pin, goes through several levels of different LABs and then reaches the output pin. MAX 7000B's maximum logic array delay for a signal to go through a macrocell is 1.7ns (t_{LAD}) [17]. A signal from an input pin to a PIA interconnect is 0.6ns (t_{IN}). The delay incurred by a signal that travels through PIA is about 2.4ns (t_{PIA}). Based on these and other timing parameters [17], we can derive that an edge delay between two different non-registered macrocell outputs from different LABs is about 4.6ns (we denote it as $t1$); [5] and the delay between a macrocell output and an output pin is about 0.8ns ($t2$). Assume that on average each macrocell borrows two sets of parallel expanders (each set contains 5 expanders), the extra delay incurred is 2 * 0.3 = 0.6ns ($t3$). As an example, the critical path of circuit alu4 under PLAmap starts from input k; goes through four LABs: Y, R, AE, and S; then reaches output r. The total delay can be calculated as $t_{IN} + (t1 + t3) * 4 + t2$. The result is 0.6 + 5.2 * 4 + 0.8 = 22.2ns, which is close to the result shown in Table 4 (23.7ns). This experiment shows one of the CPLD features, i.e., predictable timing.

5. CONCLUSIONS AND FUTURE WORK

We have presented a new performance-driven mapping algorithm, PLAmap, for CPLD structures. Our algorithm breaks the technology mapping process into three stages: labeling, mapping and packing. Our 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 structures with a large number of small PLAs such as $(10, 12, 4)$-PLAs, we compared our results with a recently-published technique TEMPLA. The comparison shows that TEMPLA uses 8 to 11% less area but about 96 to 106% more depth than PLAmap. TEMPLA also consumes huge run-time. Commercial CPLDs are usually based on large PLAs such as $(36, 80, 16)$-PLAs with special structural constraints. To demonstrate that our algorithm can also be applied to commercial CPLDs, we modified our program to take into account structural constraints in Altera's MAX 7000B CPLD structure. Experimental results show that Altera's MAX+PLUS II uses 12% less area but 58% more delay compared to PLAmap.

One direction of future work is to try other area/delay tradeoff techniques so the tradeoff can be more controlled and provide various solutions to meet different mapping requirements. We also plan to apply PLAmap to other commercial synthesis tools from major EDA companies such as XILINX, Cypress, and Lattice. Extension of PLAmap for arithmetic circuits is also under consideration.

[5]Delay between registered macrocell outputs is about 2ns more than $t1$.

6. ACKNOWLEDGMENTS

The authors gratefully acknowledge Jason Anderson for providing helpful discussions and the whole TEMPLA experiment environment. We thank Grachelle Garcia at Altera for helping us understand the specific structural constraints in the MAX 7000 and MAX 9000 CPLD families. Thanks also to David Mendel at Altera, Maureen Smerdon at Lattice, and Michel Campmas and Kuyler Neable at Cypress for kindly providing their software tools. Finally, we would like to thank Dr. Songjie Xu and Dr. Chang Wu at Aplus Design Technologies, Inc., and Dr. Wangning Long at UCLA for their helpful discussions. This research is supported in part by Altera Corp. and Lattice Semiconductor Corp. under the California MICRO program and the NSF Grants MIP-9725771 and MIP-9357582.

7. REFERENCES

[1] J.H. Anderson and S.D. Brown, "Technology Mapping for Large Complex PLDs", Proc. 35th ACM/IEEE Design Automation Conference, 1998, pp.698-703.

[2] K.C. Chen, J. Cong, Y. Ding, A. Kahng, and P. Trajmar, "DAG-Map: Graph-Based FPGA Technology Mapping

	Circuit Information for BSet_3				PLAmap BSet_3			Altera NoMLS & Normal/BSet_3		
Bench-marks	Gate No.	Input/ Output	Gate Level	Latch No.	LAB No.	MC Level	Delay (ns)	LAB No.	MC Level	Delay (ns)
planet	467	7/19	14	6	9	3	15.3	6	2	10.9
s1423	502	17/5	46	74	26	4	21.6	23	4	26.7
s1488	578	8/19	12	6	7	3	14.5	5	2	10.8
s1494	585	8/19	12	6	6	2	10.8	5	2	10.8
s9234*	933	36/37	21	135	15	3	14.5	31	4	25.8
sbc	695	40/56	11	27	18	3	15.8	15	2	12.6
scf	673	27/54	12	7	22	3	16.1	10	2	11
tbk*	682	6/3	13	5	10	4	19.4	5	5	24.6
s1238	526	14/14	16	18	11	3	15.5	8	4	21.4
s1196	492	14/14	19	18	10	3	15.2	8	4	22
s838*	261	34/1	26	32	9	3	15.4	10	13	61.6
minmax10	376	13/10	53	30	16	7	35.2	16	12	63.8
ex5p	1544	8/63	13	0	11	2	12.4	10	1	7.2
alu4*	624	14/8	29	0	16	4	23.7	11	10	49.5
t481*	580	16/1	15	0	8	3	16.7	2	4	20.6
x4	302	94/71	8	0	14	1	8.1	14	1	8.1
C3540	1017	50/22	40	0	24	6	34.3	28	11	63.6
C1355*	494	41/32	21	0	10	3	17.1	8	4	22.7
vda	445	17/38	10	0	10	2	12.3	10	2	12.4
k2	789	45/43	15	0	19	2	12.9	21	3	17.9
apex4*	2316	9/18	13	0	25	2	13.8	16	14	68.2
duke2	418	22/29	11	0	5	2	11.3	5	1	7.3
C880	357	60/26	24	0	13	3	17.6	8	7	38
table5	917	17/15	13	0	10	2	12.3	11	3	16.5
Total	-	-	-	-	324	73	401.8	286	117	634
Comparison	-	-	-	-	1	1	1	-11.7%	+60.3%	+57.8%

Table 4: Area/delay comparison of PLAmap and Altera's MAX+PLUS II. Altera's results are obtained with NoMLS style if possible. Circuits with '*' indicate that they are rerun with Normal style.

for Delay Optimization", IEEE Design and Test of Computers, Sept. 1992, pp.7-20.

[3] J. Cong and Y. Ding, "FlowMap: An Optimal Technology Mapping Algorithm for Delay Optimization in Lookup-Table Based FPGA Designs", IEEE Trans. on Computer-Aided Design, Jan. 1994, Vol. 13, No. 1, pp.1-12.

[4] J. Cong and Y. Ding, "On Area/Depth Trade-off in LUT-based FPGA Technology Mapping", IEEE Transactions on Very Large Scale Integration (VLSI) Systems, vol.2, (no.2), June 1994. pp.137-148.

[5] J. Cong and Y. Ding, "Combinational Logic Synthesis for LUT Based Field Programmable Gate Arrays", ACM Trans. on Design Automation of Electronic Systems, Vol. 1, No. 2, April 1996, pp.145-204.

[6] J. Cong and S. Xu, "Synthesis Challenges for Next-Generation High-Performance and High-Density PLDs", Asia and South Pacific Design Automation Conference, Yokohama, Japan, Jan. 2000, pp. 157-162.

[7] J. Cong, H. Huang and X. Yuan, "Technology mapping for k/m-macrocell Based FPGAs", Proc. ACM/SIGDA International Symposium on Field Programmable Gate Arrays, San Jose, CA., Feb. 2000, pp. 51-59.

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

[9] Z. Hasan, D. Harrison and M. Ciesielski, "A fast Partition Method for PLA-based FPGAs", IEEE Design and Test of Computers, Dec. 1992, pp.34-39.

[10] S.P. Khatri, R.K. Brayton, A. Sangiovanni-Vincentelli, "Cross-talk Immune VLSI Design Using a Network of PLAs Embedded in A Regular Layout Fabric", IEEE/ACM International Conference on Computer Aided Design, Nov. 2000, pp.412-418.

[11] J.L. Kouloheris and A. El Gamal, "FPGA Performance vs. Cell granularity", Proc. Custom Integrated Circuits Conference, 1991, pp.621-624.

[12] J.L. Kouloheris, "Empirical Study of the Effect of Cell Granularity on FPGA Density and Performance", PhD thesis, Stanford University, 1993.

[13] E.L. Lawler, K.N. Levitt, and J. Turner, "Module Clustering to Minimize Delay in Digital Networks", IEEE Trans. on Computers, Vol. C18(1), Jan. 1969, pp.47-57.

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

[15] Ellen Sentovich, et.al., "SIS: A System for Sequential Circuit Synthesis", Electronics Research Lab., Memo. No. UCB/ERL M92/41, 1992.

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

[17] *Understanding MAX 7000 Timing*, Application Note 94, Altera Corporation, May 1999.

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

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