Technology Mapping Algorithms for Hybrid FPGAs Containing Lookup Tables and PLAs

Srini Krishnamoorthy, *Student Member, IEEE*, and Russell Tessier, *Member, IEEE*

***Abstract*—Programmable devices containing lookup tables (LUTs) and programmable logic arrays (PLAs) provide a heterogeneous target platform for user designs. Present commercial tools, which target these hybrid devices, require hand partitioning of user designs to isolate logic for each type of logic resource. In this paper, an automated technology mapping tool, *hybridmap*, is presented that identifies design logic partitions as suitable for either LUT or PLA implementation. A breadth-first search-based subgraph extraction and evaluation heuristic is integrated with product term (Pterm) count, area, and delay estimators to guide the technology mapping process. *Hybridmap* can be adapted to target a variety of PLA architectures and can accommodate user-provided timing constraints. It is shown that when timing constrained, *hybridmap* reduces LUT consumption for Apex20KE devices (Altera Corporation 1999) by 8% and when unconstrained by 14% by migrating logic from LUTs to Pterm structures. *Hybridmap* is shown to outperform previous mapping approaches (Lin and Wilton 2001) for Apex20KE-type devices by up to 22%.**

***Index Terms*—Hybrid field-programmable gate array (FPGA), lookup table (LUT), programmable logic arrays (PLAs), technology mapping.**

I. Introduction

RECENT innovations in field-programmable gate array (FPGA) architecture have led to the development of *hybrid* FPGA families [3] that combine diverse sets of logic resources on the same silicon substrate. To support wide-fanin, low logic-density subcircuits, such as finite-state machines, some contemporary FPGA architectures [4] contain SRAM-configurable programmable logic arrays (PLAs). Unlike fine-grained lookup tables (LUTs), PLAs can implement sets of logic functions with minimal interconnect, the most area-expensive resource in contemporary FPGAs [3]. For product term (Pterm)-based PLA structures, this area efficiency often comes at the cost of increased minimum delay for PLA paths versus corresponding LUT paths, requiring resource balance. When coupled with fine-grained LUTs, PLAs provide an integrated programmable resource that can be used in many digital system designs to support noncritical-path control logic for LUT-based datapaths. Current industry technology mapping tools [5], [6] do not provide automated techniques to partition user designs across heterogeneous logic resources, limiting the usefulness of hybrid devices. This work presents an automated technology mapping tool, *hybridmap*, that automatically partitions user designs to a collection of LUTs and PLAs so that an area-optimized solution is achieved. This involves packing as much logic as possible into available PLAs, thus minimizing required LUTs.

As shown in Fig. 1, contemporary FPGA architectures generally contain highly optimized blocks which include both logic and routing resources. These blocks are replicated in a vendor-specific pattern throughout the device. A coarse-grained structure, such as an embedded PLA, is allocated one per m fine-grained LUTs. Hierarchical routing resources provide required connectivity among nonadjacent device structures.

Our new *hybridmap* tool automates PLA logic extraction and subsequent PLA and LUT mapping. Since FPGA devices generally contain proportionally more LUT than PLA resources, design subgraphs, initially targeted at LUTs, must be retargeted at PLAs. As a result, subgraph resource estimation forms a significant part of our approach. The developed system integrates a series of new graph search heuristics with novel cost functions to identify subgraphs quickly. *Hybridmap* can be run to target two distinct objectives: area minimization without regard to design performance and timing-constrained area minimization. The area to be minimized is defined in terms of the post-mapping K-LUT count required to implement the design. When the technology mapping objective is unconstrained area minimization, *hybridmap* attempts to minimize LUT count by packing as much logic as possible into PLAs. When a timing constraint is specified, *hybridmap* controls the PLA packing process so that LUT count is minimized subject to prespecified timing constraints.

Hybrid device mapping takes place through a series of interrelated steps. Input to *hybridmap* is represented as a directed acyclic graph (DAG). Following preprocessing, a breadth-first search algorithm generates logic subgraphs satisfying the input, output, and Pterm constraints of the target PLA. An estimator of the number of Pterms required by a subgraph (Pterm count estimator) determines if a candidate subgraph meets the Pterm constraint of the target PLA. Unlike minimization approaches, such as *Espresso* [7], the estimator is sufficiently fast ($<<$ sec) to allow embedding within the inner loop of subgraph extraction. Following extraction, candidate PLA subgraphs are ranked based on LUT coverage and mapped to available PLAs. The logic not mapped to PLAs is implemented in LUTs. When mapping under timing constraints, a delay estimator is integrated into the design flow to evaluate the effect of subgraph extraction on design performance.

To illustrate the effectiveness of *hybridmap*, the tool has been used to map a set of the Microelectronics Center of North Carolina (MCNC) [8] benchmark circuits to hybrid devices. Results

Manuscript received January 10, 2002; revised April 8, 2002 and July 16, 2002. This work was supported in part by the National Science Foundation under Grant CCR-0081405 and in part by a grant from the Xilinx Corporation. This paper was recommended by Associate Editor M. Sarrafzadeh.

The authors are with the Department of Electrical and Computer Engineering, University of Massachusetts, Amherst, MA 01003 USA (e-mail: sikrishn@ecs.umass.edu; tessier@ecs.umass.edu).

Digital Object Identifier 10.1109/TCAD.2003.810743

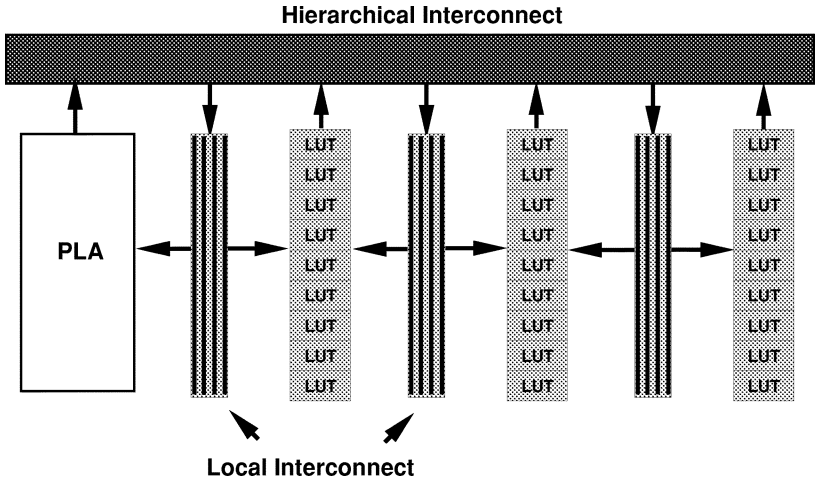


Fig. 1. Hybrid FPGA block (similar to Apex20KE [1]).

were obtained by mapping to Altera's Apex20KE devices [1] for both unconstrained and timing-constrained area minimization. When mapping under timing constraints, *hybridmap* reduces required LUTs by 8% by mapping covered logic to PLAs. This value increases to 14% if timing constraints are not considered. This allows a larger design to be packed into a specific device or the same design to be packed into a smaller, less costly device.

In Section II, a description of the background material for hybrid technology mapping is presented. Section III motivates our approach through the analysis of circuit data. In Section IV, our technology mapping approach for unconstrained mapping is described, while in Section V, timing-constrained mapping is discussed. Details of the target FPGA architecture (the Altera Apex20KE), are described in Section VI. Experimental results obtained by applying *hybridmap* to a collection of benchmark circuits are presented in Section VII. Finally, Section VIII summarizes our research and outlines directions for future work.

II. Background

A. Problem Definition

A hybrid LUT/PLA FPGA device consists of a collection of K-input LUTs and multi-input, multi-output PLAs. The Pterm-based PLA resource supports i_m inputs, o_m outputs, and p_m Pterms. For PLAs, i_m, and o_m define the PLA structural constraint and p_m defines the PLA functional constraint.

For a target device containing r PLAs, each of which can be configured with i_m inputs, o_m outputs, and p_m Pterms, the technology mapping objectives of unconstrained area minimization and timing-constrained area minimization can be defined as follows:

- *Unconstrained area minimization.* Given an input circuit, locate a mapping to r circuit subgraphs and q K-input LUTs such that input, output, and logic constraints of corresponding resources are satisfied, the number of post-PLA mapping LUTs, q, is minimized, and q is less than N_{LUTs}, the number of LUTs available in the device.
- *Timing-constrained area minimization.* Given an input circuit locate a mapping to r circuit subgraphs and q K-input LUTs such that input, output, and logic constraints of corresponding resources are satisfied, the number of post-PLA mapping LUTs, q, is minimized, and q is less than N_{LUTs}, the number of LUTs available in the device. The mapped circuit should operate at a minimum clock frequency F.

Theorem 1: The unconstrained area minimization problem for hybrid FPGAs containing a bounded number of LUT and PLA resources is NP-complete for general networks.

Proof: It has been shown that the area-optimal technology mapping problem for K-bounded networks is NP-complete [9]. The subgraph extraction problem is more general than the K-bounded, single-output problem since multi-output subgraphs can consist of a collection of K-bounded, single-output networks. Therefore, unconstrained area minimization is NP-complete. □

Theorem 2: The timing-constrained area minimization problem for hybrid FPGAs containing a bounded number of LUT and PLA resources is NP-hard for general networks.

Proof: It has been shown that delay-bounded technology mapping for heterogeneous FPGAs with LUTs and memory blocks is NP-hard [10]. Since subgraph implementation in a PLA block is more restrictive than subgraph implementation in a memory block, the timing-constrained area minimization problem for FPGAs with LUT and PLA resources is also NP-hard. □

B. Terminology

Input to *hybridmap* is a combinational circuit represented as a DAG, $G(V, E)$, containing combinational nodes, V, and interconnection edges, E. Each node is a complex gate implementing a local function as a sum-of-products representation of its input signals. For a given node v, $\mathrm{fanin}(v)$ is the set of nodes that drive v and $\mathrm{fanout}(v)$ is the set of nodes driven by v. The cone of a node v, $\mathrm{cone}(v)$, is the set of transitive fanins of v. The depth (d_v) of a node v is the length of the longest path, in terms of available LUT resources, from primary inputs to v. The required signal arrival time $RT(v)$ is the time point a signal is needed at the input or output to a node v. The slack-value, $SV(v)$, of a node v is the difference between the required signal arrival time

at the output of v and the depth of v. For a specific hybrid device, a subgraph is considered *PLA-feasible* if the input, output, and Pterm count of the subgraph satisfies the constraints of the target PLA resource.

C. Related Work

Logic synthesis targeting FPGAs has been researched extensively and numerous technology mapping approaches for LUT-based FPGAs [11] have been developed. These approaches have two main objectives, area and delay minimization, and can be characterized by input representation. Input network types include tree-type [12], [9], MFFC-type [13], and general networks [14], [15]. To address delay minimization issues, delay models such as the unit-delay model [16], net-delay model [17], and edge delay model [18] have been proposed.

To date, most research in FPGA technology mapping has focused on FPGAs containing homogeneous type of resources, although recently, technology mapping algorithms for devices with LUTs of differing input sizes [19], [20] and PLAs have been presented. In [21], a technology mapping algorithm for devices with K-input, single-output macrocells was presented. The algorithm determines the minimum height K-feasible cut for circuit nodes and their cones. A heuristic technique is described that exhaustively enumerates Pterm mapping options and generates area and delay efficient design implementations. Another technique [22], based on *dag-map* [14], minimizes delay for macrocell architecture mapping. Since this architecture contains a homogeneous set of logic resources, it is not necessary to consider tradeoffs between resources with different mapping qualities (e.g., low versus high fanin, logic density) in making mapping decisions.

The introduction of coarse-grained memory elements in commercial devices (e.g., Altera's Flex10K [23] and Xilinx's Virtex [24]) has motivated graph search approaches that identify suitable logic partitions for implementation in unused block memories. Technology mapping approaches, described in [10], [25], and [26], configure unused embedded memory blocks as large multi-output ROMs to increase device utilization. These memory packing algorithms provide insight into the hybrid LUT/PLA FPGA mapping problem by identifying portions of an input logic design that are appropriate for implementation in a restricted resource. While memory blocks that have not been used to implement memory functions can be leveraged to implement combinational functions with extended logical depth, limited memory-input counts currently restrict the breadth of logic functions that can be implemented. As a result, wide-fanin subcircuits, such as finite-state machines, must be migrated to device LUTs.

Recently, Kaviani [27] investigated both the architectural parameters of hybrid FPGA architectures and supporting technology mapping approaches. The described technology mapping approach [27] for hybrid LUT/PLA architectures applies partial collapsing and partitioning to isolate wide-fanin logic nodes with single outputs. Input sharing is then used to determine the nodes to be merged into a PLA. The target devices in our work contain wide-fanout PLAs, which are structures with more than three outputs, necessitating subgraph-based approaches rather than node-based approaches.

Lin and Wilton [2] developed a technology mapping algorithm targeting hybrid architectures. Input to the PLA-mapping algorithm is first pre-mapped to four-LUTs using LUT technology mapping tools [15]. Subsequently, for each node v in the mapped circuit, a local graph search collects transitive fanins of v such that the overall input, output and Pterm count of the node set satisfies the PLA constraints. The collected set of nodes is not fanout-free and the intermediate nodes that drive nodes outside of the collected node set are implemented as subgraph outputs. Each subgraph is subsequently mapped to PLAs. The remainder of the design that is not mapped to PLAs is the LUT partition. Since the PLA logic extraction approach is localized, the algorithm is unable to identify and cover reconvergent paths. In this paper, a hill-climbing phase is performed during subgraph generation to cover reconvergent paths with PLA logic. A preliminary version of our hybrid technology mapping approach was previously presented [28] which did not provide for this extended search or for timing-constrained mapping.

III. Methodology Motivation

Previous approaches to hybrid mapping have focused on *node-collapsing* techniques where multiple single-output, fanout-free cones are merged into a multi-output node. This technique can be used effectively to map subcircuits to PLAs with small numbers of outputs [27]. For wide-fanout PLAs, *graph-based* combinational node search approaches are needed. Subgraph logic, mapped to Pterms, ideally share a significant number of inputs. Several procedures used by our advanced *hybridmap* approach are motivated based on subgraph and design statistics.

As the number of PLA outputs increases, it would seem likely that opportunities for Pterm input sharing across outputs would also increase. The solid plot in Fig. 2 indicates that this is indeed the case. The figure shows that as the number of outputs per PLA grows, the average number of Pterm-based subgraph outputs driven by each input increases. Data points for the graph were collected from 1000 subgraphs extracted from a collection of MCNC [8] benchmarks discussed in Section VII. For low PLA output count values, little input sharing is present, but as the number of PLA outputs increases, sharing becomes more prevalent. Our experiments show that circuitry targeted to PLAs should be identified via subgraph identification rather than first isolating single-output logic cones followed by cone merging to obtain multiple outputs. The dashed plot in Fig. 2 furthers this assessment. The plot shows the average number of outputs driven by each Pterm in a subgraph grows as a function of subgraph output count. Shared output values greater than one indicate Pterm sharing among multiple outputs.

Unlike subgraph input and output counts, subgraph Pterm counts can be difficult to estimate quickly. Subgraphs with Pterm counts which exceed PLA constraints can be eliminated from mapping consideration. Fig. 3 shows the number of subgraphs $N(P_i)$ requiring $Pterm\ counts < P_i$ prior to *and after* subgraph logic minimization. A total of 1800 subgraphs were considered where each of the subgraphs contains a maximum of 32 inputs and 16 outputs to match a target PLA architecture of 32 inputs and 16 outputs. The figure illustrates that the

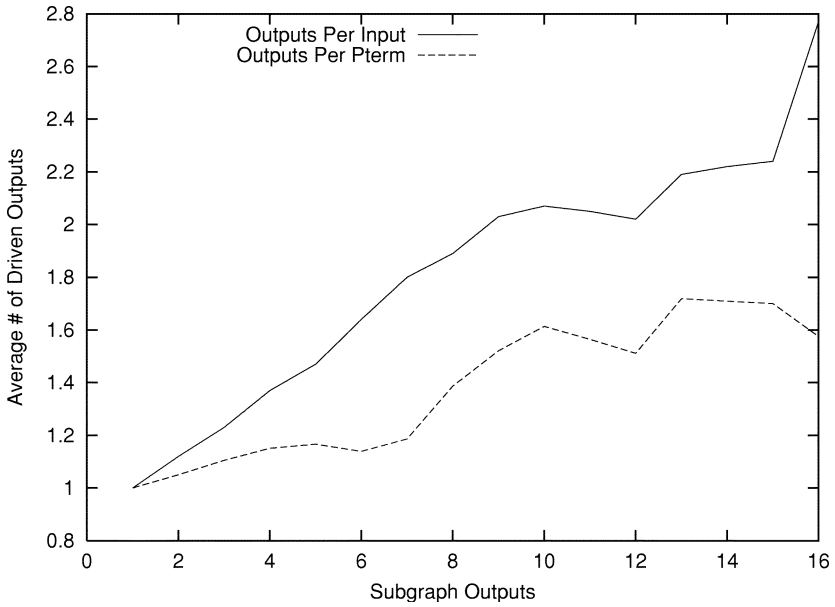


Fig. 2. Average number of outputs driven by a PLA subgraph input and Pterm.

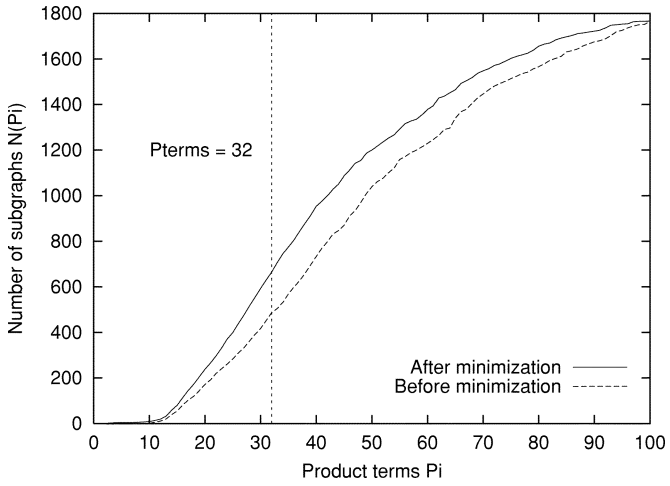


Fig. 3. Number of subgraphs meeting Pterm count P_i.

number of Pterm-feasible subgraphs (e.g., $Pterm\ count \leq 32$) identified *prior* to logic minimization is relatively small (about 28%). As shown in Fig. 3, experimentation with Espresso [7] indicates that 10% more subgraphs become PLA-feasible following logic minimization. The availability of a larger number of available PLA-feasible subgraphs provides greater flexibility in choosing PLA partitions that provide a maximal reduction in post-mapped LUT count. This finding motivates our development of a fast *Pterm count estimator* for subgraphs which can estimate post-minimization Pterm counts without performing exhaustive logic minimization.

IV. Methodology for Unconstrained Area Minimization

A. Software Overview

As shown in Fig. 4, *hybridmap*, targeting area minimization without timing constraints, uses a collection of heuristics to perform hybrid technology mapping. Input circuits are represented in gate-level form (.blif format) as a forest of trees composed of combinational nodes. Each node in a tree represents a logic function in sum-of-products form. Specific processing steps include the following.

- **Design preprocessing**. Input circuit logic is initially reduced using *SIS* [29] technology independent optimization scripts *script.algebraic* or *script.rugged*. Resulting complex gates are then decomposed to two-input gate form using Huffman-tree decomposition script *dmig* [15] to facilitate high density PLA packing.
- **LUT identification and subgraph extraction and merging**. *Hybridmap* performs cone-based clustering on the two-input gate representation to heuristically identify K-input, one-output LUTs. Following LUT identification, subgraph generation builds an initial set of PLA-feasible subgraphs and subgraph merging combines subgraphs to maximize logic coverage in PLAs. These phases integrate a subgraph search algorithm, which identifies reconvergent paths, and new Pterm count and area estimators to quickly evaluate subgraph fitness.
- **Subgraph selection**. After completing subgraph generation and merging, subgraphs are ranked based on the number of K-LUTs covered by each subgraph. The r subgraphs with the highest rank are packed into the r available PLAs while the rest of the user design is marked as the K-LUT partition.
- **Vendor-specific computer-aided design (CAD)**. The design partitions are output in hierarchical (two-level) VHSIC hardware description language format and presented to vendor synthesis and place and route tools for mapping to the target architecture.

B. LUT Identification

After technology independent optimization and decomposition, *hybridmap* combines sets of two-input gates into K-input, one-output nodes using a dual-phase approach from *dag-map*

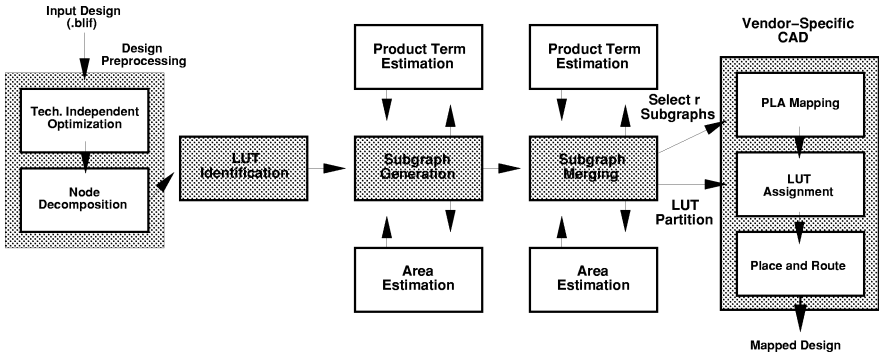


Fig. 4. Unconstrained area-minimization mapping flow.

[14]. A depth assignment phase identifies the depth (d_v) of each graph node v in terms of K-LUT delay values ($\mathrm{delay_{LUT}}$). This assignment is made via a search performed in topological order from circuit primary inputs to primary outputs [14]. Following depth assignment, nodes are clustered based on their node depth values. Clusters are formed via a search performed from circuit primary outputs to primary inputs [14]. During subsequent subgraph generation, the clusters are used for area and depth estimation purposes only. The original, unclustered graph is used for the subgraph search.

C. Subgraph Generation

The subgraph generation process identifies input- and output-feasible subgraphs from the input graph G using a localized graph search approach. Input–output (I/O) feasible subgraphs are combinations of nodes which meet the structural constraints of the PLA. For a selected node v in graph G, a forward traversal phase collects a tree consisting of transitive $\mathrm{fanouts}(v)$ and identifies a set of nodes RS driving no more than o_m leaves, where o_m is the available output count of the PLA resource. During a second phase, an inverse search traverses transitive fanin nodes of RS to identify subgraph input signals and the nodes that can be absorbed into the PLA. Following each subgraph identification, a Pterm count estimator evaluates the number of Pterms required to implement the subgraph.

1) Basic Approach: As shown in Fig. 5, a subgraph is identified by starting at a node v, selected in topological order from primary inputs to primary outputs. During a forward traversal phase, starting from v, the transitive $\mathrm{fanouts}(v)$ are visited iteratively in a breadth-first fashion to identify a tree T rooted at v. At each iteration step, a new set of tree leaves is identified. In Fig. 5(a), for a selected node v, the shaded nodes form the new leaves after the second iteration. New leaves are added in a breadth-first fashion until their number exceeds the output constraints of the target PLA. At the end of the forward search, the leaves of tree T are designated as the root set, RS, of the traversal. In Fig. 5(a), the shaded leaf nodes form the root set of the traversal starting at v.

Once the root set has been determined, an inverse traversal of G iteratively collects transitive fanins of the root set in a breadth-first fashion to determine the subgraph associated with the root set. At each iterative step in the basic approach, fanin nodes to the root set are included only if their fanout is limited to

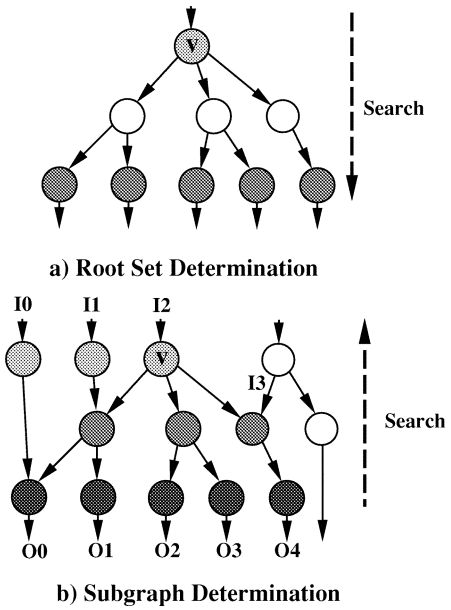


Fig. 5. Root set and subgraph determination.

the current subgraph. Collection of fanin nodes continues until the subgraph input count exceeds the input count of the PLA resource (i_m). As shown in Fig. 5(b), subgraphs constructed in this way, that meet the allowed Pterm count, can be targeted to a PLA since their input and output counts are guaranteed to fit PLA constraints. In order to avoid logic duplication and to improve the runtime, seed nodes for subgraph generation are selected only from the nodes that are not already covered by any other subgraph. Additionally, the subgraphs are mutually exclusive and include only those nodes not covered by other subgraphs. Pseudocode for the subgraph generation algorithm is shown in Fig. 6.

2) Hill Climbing: Taking Advantage of Reconvergent Paths: The basic subgraph generation approach can be augmented to take advantage of the reconvergent nature of some logic. Reconvergent paths originate either as a design artifact or due to recursive logic decomposition and resubstitution operations performed during the technology independent optimization process. These paths begin from a set of graph nodes $s1$ (e.g., $|s1| < i_m$), diverge to drive inputs of multiple nodes (set $s2$ where $|s2| > |s1|$, e.g., $|s2| > i_m$ and o_m), and converge down to a set of nodes (set $s3$ where $|s3| < |s2|$, e.g., $|s3| < o_m$). Fig. 7 shows a circuit where output signals

```
Input: Combinational circuit graph, G.
Output: Forest of subgraphs, N

For each node v in G.
    Traverse G from node v along fan-out edges
        to locate root set RS with o_s ≤ o_m.
    Traverse G from RS along fan-in edges to
        determine Subgraph with i_s ≤ i_m.
    Estimate Pterms p_s in Subgraph
    If p_s ≤ p_m
        Add Subgraph to N
    EndIf
endFor
```

Fig. 6. Pseudocode for the subgraph generation algorithm.

from four gates ($|s1| = 4$) diverge as inputs to multiple gates ($|s2| = 5$) before converging into two outputs ($|s3| = 2$). These reconvergent paths could be covered with a single PLA subgraph boosting LUT savings. Alternative PLA covering approaches [2], [22], [27] do not identify reconvergent paths that exceed PLA I/O limits. In our approach, a hill-climbing phase extends forward and inverse traversal beyond the point where PLA input and output limits are exceeded with the expectation that the subgraph will reconverge.

The modified graph search approach starts in the basic subgraph generation mode. During either the forward or inverse traversal phase of the search, the first instance of a PLA output or input constraint violation forces the graph search to the hill-climbing mode. If, at a further point in the graph search, the I/O counts of the subgraph meet PLA i_m, o_m constraints, the graph search switches back to the basic approach of subgraph generation. This second phase of the basic subgraph generation algorithm terminates whenever PLA input or output count violations are observed. Fig. 7 shows the hill climbing phase applied during the forward traversal phase when targeting a four-input, two-output PLA. During experimentation, hill climbing was found to improve LUT coverage versus the basic approach for about 10% of searches.

3) Subgraph Pruning: If the hill-climbing procedure never finds input or output counts that meet the PLA constraints it will terminate upon reaching a fixed search depth, L, or circuit primary inputs, primary outputs or flip-flop inputs. As a result, subgraphs generated using hill climbing may violate PLA input, output or Pterm constraints. These subgraphs can be pruned to fit in a PLA by iteratively removing excess inputs and outputs. As an initial step of pruning, subgraph logic is collapsed to a two-level representation (sum-of-products form) and subgraph outputs are ranked in nondecreasing order by input requirements. Each output requiring less than K inputs can be implemented using a single K-LUT. As this represents a minimal penalty in terms of LUT coverage, logic and inputs solely associated with these outputs are removed first from the subgraph. This is followed by minimal multi-LUT removal, if necessary, until structural constraints are met.

The approach is optimal when the paths that are followed in determining a subgraph are the only possible paths.

Lemma 1: In the worst case, subgraph generation is performed over all nodes V and each node is visited during each iteration of subgraph generation. This search results in a time complexity of $O(V^2)$.

D. Product Term Count Estimation

During the subgraph generation process, each subgraph satisfying the input and output constraints of the PLA is evaluated as a potential candidate for PLA mapping. Subgraphs are collapsed into two-level form for direct Pterm count verification against the PLA Pterm count constraint.

Since Pterm count must be evaluated for each subgraph, the Pterm count estimator runtime impacts the usability of the estimator. Specifically, the estimator needs to satisfy the following requirements: 1) it has to be fast, requiring runtimes well under a second and 2) it only needs to verify that the post-minimization Pterm count of a subgraph is less than that allowed by the PLA. In order to quickly determine the post-minimization Pterm count, three Pterm count estimators were considered: a statistical technique, Espresso [7], and a new Pterm estimator.

1) Statistical Approaches: The statistical estimator attempts to predict the post-minimization Pterm count based on initial preminimization subgraph Pterm counts. To explore the usefulness of statistical data, 1000 wide-fanin ($i_s \leq 32$), wide-fanout ($o_s \leq 16$) subgraphs extracted from MCNC [8] benchmark circuits described in Section VII were passed through the logic minimization tool, Espresso. It was observed that for a given input and output count ($i_s \leq 32$, $o_s \leq 16$) the average Pterm count requirement per subgraph prior to and after minimization was 50 with a standard deviation > 10 and 29 with standard deviation > 9 respectively. For the above-mentioned subgraphs, the most important metric, the Pterm count reduction per subgraph, had a standard deviation of 10, making post-minimization Pterm count prediction impossible.

2) Espresso: Espresso finds a minimal Pterm count cover for each subgraph output f_i in a iterative manner. During each step, each Pterm (cube) is expanded to explore the possibility of covering other Pterms (cubes) in the representation. The covered Pterms are subsequently removed. Exhaustive enumeration steps and the exact minimization operations in Espresso are characterized by long runtimes (e.g., 10 min) for wide-fanin ($i_s \leq 32$), wide-fanout ($o_s \leq 16$) subgraphs. The optimal Espresso option *opoall* attempts to minimize the Pterm count by exploring all 2^n patterns for an n-output subgraph F. The fastest multi-output minimization option in Espresso, *opo*, minimizes the function $F_{\text{opo}} = \{F, \overline{F}\}$ where $\overline{F}$ is the set of complemented functions $\overline{f_i}$ and selects the minimal form.

3) New Estimation Heuristic: Our new Pterm count estimator determines Pterm count by attempting to cover a cube by a maximum of two other cubes. Fig. 8(a)–(c) demonstrate the minimization steps carried out by the estimator. The input to the estimator is a two-level representation of the subgraph logic functions presented in SIS PLA format [29]. Each row indicates a Pterm (or cube) with true (1), complemented (0), or don't care (-) conditions for the input signals and each output column represents a single output. When performing estimation, only a single cube literal is expanded at a time. Minimization operations such as Pterm covering, sharing, and input expansion

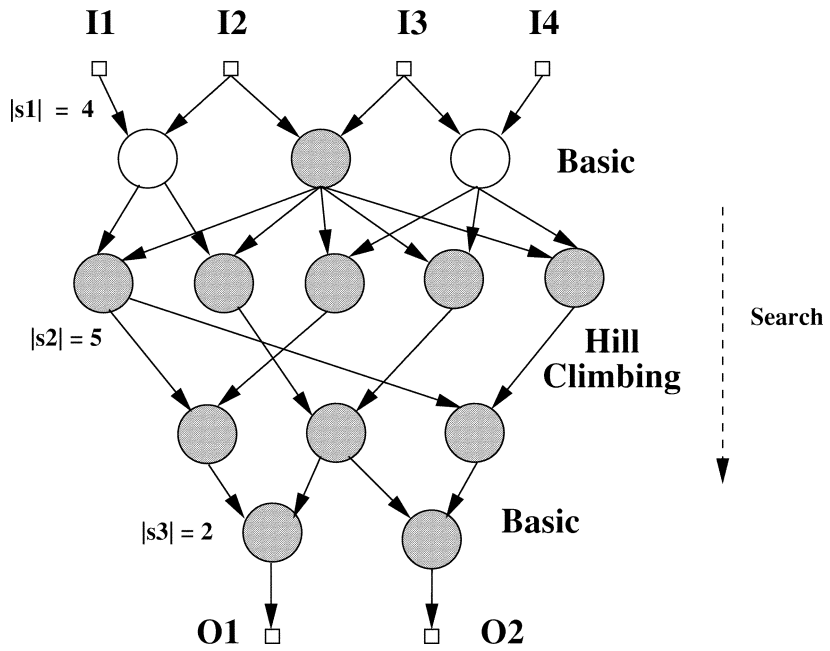


Fig. 7. Reconvergent paths in a circuit graph during forward traversal.

Before expansion		After expansion	
Input	Output	Input	Output
-101-0	1	-101-0	1
0101-0	1		
-10100	1		

(a) Example 1 - Input Expansion

Before expansion		After expansion	
Input	Output	Input	Output
101010	100	101010	100
101-10	010	101010	010
101110	001	101110	010
		101110	001

(b) Example 2 - Cube Covering (Step1)

Intermed. Result		Final Result	
Input	Output	Input	Output
101010	110	101010	110
101110	010	101110	011
101110	001		

(c) Example 2 - Cube Covering (Step2)

Before complement		After complement	
Input	Output	Input	Output
--0---	10	-01-0-	11
-1----	10		
----1-	10		
-01-0-	01		
phase	11	phase	01

(d) Example 3 - Output Complementation

Fig. 8. Minimization examples.

are applied incrementally to reduce Pterm count. In Fig. 8(a), for the given output, the first and the second Pterm differ only at the first input. As a result, the first Pterm covers the second one. Similarly, the first Pterm covers the third Pterm, reducing the overall Pterm count to one. Fig. 8(b) illustrates the covering of a single Pterm by two other Pterms. An initial Pterm count of three is required by the representation shown in the *Before Expansion* column. The second Pterm under this column can be expanded at the fourth input to form the middle two cubes under the *After Expansion* column. The first two Pterms under the *After Expansion* column can then be merged to produce the representation shown under the *Intermediate Result* column in Fig. 8(c). The final representation is obtained by merging the second and third Pterms in the *Intermediate Result*. As seen under the *Final Result* column, the Pterm count is reduced to two.

For our new estimation approach, only n spaces are considered for an n-output function, corresponding to n complementations, one output at a time. Each output is complemented starting from the one driven by the most Pterms and ending with the one driven by the fewest Pterms. An example of output complementation is shown in Fig. 8(d). By choosing to complement only the first output, the logic expressed by the first three Pterms is now covered by the fourth. The phase information below the truth table indicates whether the outputs are represented in true (1) or complemented (0) form.

Lemma 2: Given that each Pterm must be evaluated against every Pterm, the time complexity per subgraph of the estimator is $O(p^2)$, where p is the initial number of subgraph Pterms.

4) Estimation Comparison: In order to evaluate the efficiency of the Pterm estimator used in *hybridmap*, wide-fanin and wide-fanout subgraphs ($i_s \leq 32$ and $o_s \leq 16$) were

TABLE I
COMPARISON OF ESPRESSO AND ESTIMATOR FOR SUBGRAPHS WITH $i_s \leq 32$, $o_s \leq 16$

Ckt Name	Avg. Orig Pterms	*Espresso -opoall* Avg. Final Pterms	*Espresso -opoall* Time (s)	*Espresso -opo* Avg. Final Pterms	*Espresso -opo* Time (s)	*Espresso -opo* Maximum Error (Pterms)	Estimator Avg. Final Pterms	Estimator Time (s)	Estimator Maximum Error (Pterms)
apex1	43.0	26.0	594	26.1	2.80	1	26.2	0.01	1
apex3	41.0	22.0	316	22.4	0.02	3	22.7	0.01	3
apex4	39.5	25.1	582	25.3	0.06	5	25.3	0.01	3
s1196	51.3	41.3	595	42.6	0.55	3	41.5	0.01	3
c5315	61.4	36.5	610	37.3	2.30	4	37.4	0.40	4
des	64.0	57.2	99	57.2	0.75	0	57.2	0.03	0
duke2	36.8	20.8	1700	20.8	0.04	1	20.8	0.01	0
i10	63.5	50.3	581	50.4	0.30	1	51.2	0.01	4
rot	37.6	37.6	714	38.8	0.04	4	38.8	0.01	4
s298	40.5	31.5	560	31.5	0.02	0	31.5	0.01	0
Avg.	53.2	38.7	706	39.2	0.76	-	39.2	0.06	-

extracted from the MCNC benchmarks [8] listed in Table I (15 subgraphs per benchmark). Subsequently, minimization results were obtained using Espresso (with options *opoall* and *opo*) and the new Pterm count estimator. As can be seen from Table I, although *opoall* generates exact post-minimized Pterm counts, its runtime is prohibitively large to be of use. The option *opo* achieves a 99% accuracy in post-minimization Pterm count estimation in less than 1% of the runtime of *opoall*. The Pterm estimator is the fastest of all the approaches generating a 99% accurate result compared to the option *opoall* in about 8% of the time required by the option *opo*.

E. Area Estimation

To pack PLAs with subgraphs leading to maximal LUT count reduction, candidate subgraphs are ranked based on their LUT coverage. Our area estimator determines post-mapping LUT reduction due to each subgraph based on the following considerations: 1) Each primary output (PO) or flip-flop input in the input graph is an output of a LUT or a PLA and 2) Except for primary inputs, each LUT or PLA input is also a LUT or PLA output. The LUT identification process in Section IV-B iteratively computes the minimal LUT depth of each node to identify LUT boundaries. A change in LUT depth along an input-output path implies introduction of a new LUT. Nodes with the same depth can be collapsed into a single LUT subject to LUT I/O constraints. The area estimator uses the following strategy: A LUT output is identified at locations where the depth (d_v) of a node v is less than the depth of at least one $\mathrm{fanout}(v)$.

Mathematically, the condition for a node to be the output (C_o) of a LUT (C) can be stated as

$$v = C_o \text{ if } v \in PO \text{ or } d_v < d_k | \exists k \in \mathrm{fanout}(v). \quad (1)$$

When counting the number of LUTs covered by a subgraph, each intermediate node v, with depth d_v, satisfying (1) is counted as the output to a K-LUT and the LUT count is incremented by one. Additionally, the input side boundary of the subgraph is evaluated for any LUT count penalty. If a subgraph input node i_s, with depth d_{i_S}, is not already the output to an existing LUT cluster, an extra LUT that generates i_s is required. For every input node i_s that does not satisfy (1), the LUT count covered by the subgraph is decremented by one to account for the increase in post-mapping LUT count.

Fig. 9(a) shows a circuit covered by nine three-LUT clusters. Intermediate nodes $\{n1, \ldots, n8\}$ and $O1, O2$ cover a subgraph with inputs I0, I1, I2, I3, and I4 and outputs O1 and O2. The numbers shown next to each node indicate the depth of the node. When the subgraph is mapped to a PLA, as shown in Fig. 9(b), a total of *seven* three-LUTs are covered collectively by the nodes $n1, n2, n3 + n6, n4 + n7, n5 + n8$ and O1, O2. I0 and I2 need to be generated to drive the PLA and, hence, the subgraph LUT coverage is decremented by two. As a result, the three-LUT coverage by the subgraph is five. The final LUT count is computed as $9 - 5 = 4$ which is equal to the post-mapping three-LUT count shown in Fig. 9(b).

Lemma 3: The time complexity of each invocation of the area estimator is $O(V)$ since in the worst case all nodes could be examined during each invocation.

Post-processing by LUT mapping tools, such as Flowmap [15], pack LUTs densely, reducing final LUT counts by about 15% on average. Since both subgraphs and circuit graphs experience the same percentage reduction during post-processing, the area estimator computes the percentage of LUT count covered by each subgraph.

F. Subgraph Merging and Ranking

Following generation, smaller subgraphs can be bin-packed based on input sharing to construct combined implementations that meet PLA i_m, o_m, and Pterm requirements.

The general gain function for merging two subgraphs j, k is given as

$$\mathrm{Gain}_{jk} = \mathrm{feas}(j,k) \times \mathrm{Area}_S \quad (2)$$

where Area_S is the estimate of the K-LUT count (obtained as described in Section IV-E) covered by the subgraphs considered for merging. A merged subgraph is judged to be *PLA-feasible* ($\mathrm{feas}(j,k) = 1$) if input, output, and post-minimization Pterm counts meet PLA requirements. Violation of PLA constraints sets $\mathrm{feas}(j,k)$ to zero. Although input and output limits can be evaluated by simple counting, Pterm count verification may require additional invocation of the Pterm count estimator. Given r target PLAs, following merging, the r feasible subgraphs that

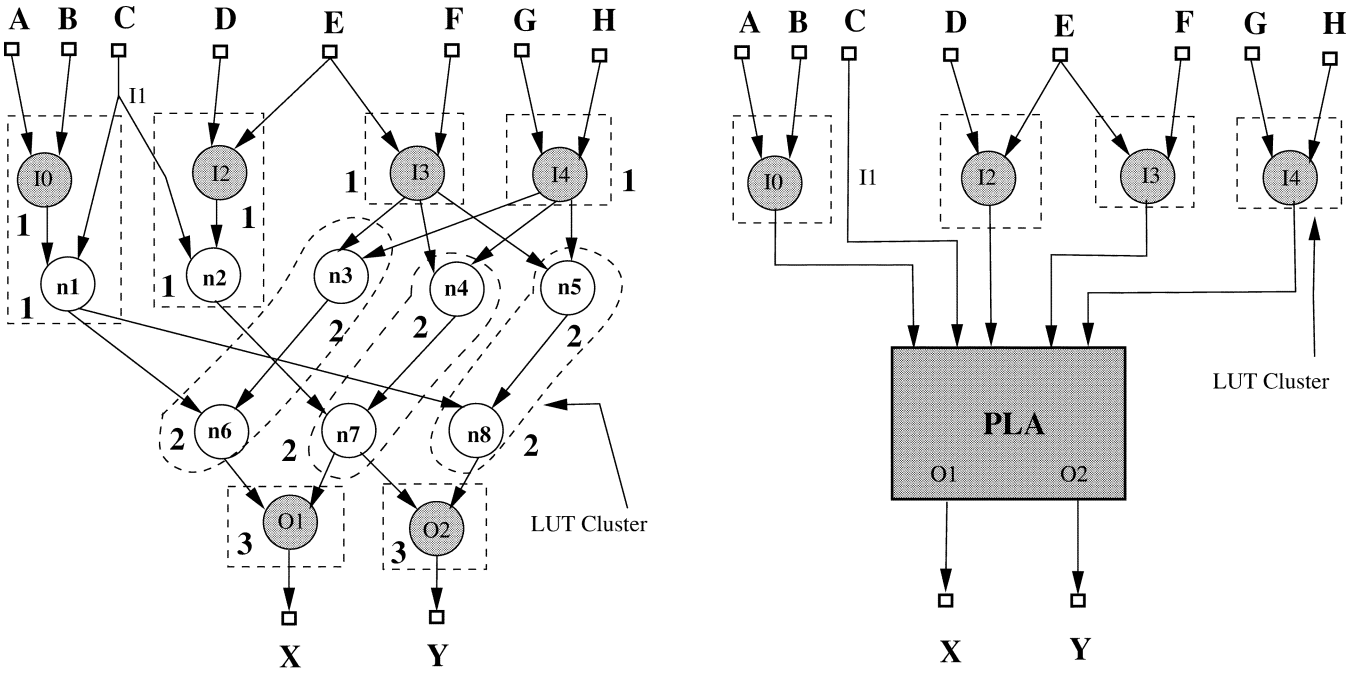


Fig. 9. Estimating LUT area covered by a subgraph. (a) Circuit covered by nine 3-LUT clusters. (b) Circuit covered by four 3-LUTs and a PLA.

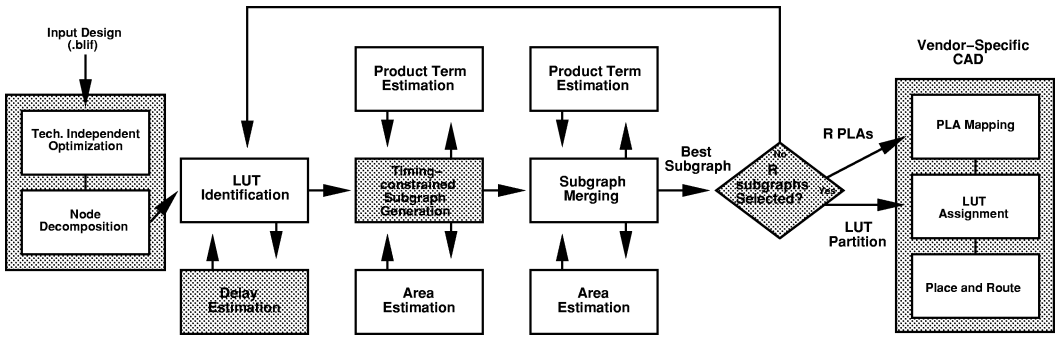


Fig. 10. Timing-constrained area minimization mapping flow.

cover the most LUTs are selected. The remainder of the circuitry is mapped to K-LUTs.

Lemma 4: The subgraph merging process has time complexity $O(V^2)$ since it can involve $O(V)$ individual comparisons for each of at most V possible subgraphs.

V. Timing-Constrained Area Minimization

Timing-constrained area minimization is invoked when a minimum design clock frequency is specified in conjunction with the design. This timing constraint must be met by the delay of the longest combinational path in the circuit when mapped to a LUT-based device. For the designs evaluated in this paper, it is determined that LUT-only mappings achieve the desired minimum frequency and that logic migration to PLAs will maintain rather than improve design performance. Since wide-fanin, wide-fanout PLAs typically incur longer delays than LUTs (e.g., $\text{delay}_{\text{PLA}} \geq 3 \times \text{delay}_{\text{LUT}}$ in Apex20KE), *hybridmap* maps noncritical paths of the original design to PLAs and the remainder to LUTs, keeping the delay of the longest combinational path within specified limits achieved by LUT-only mappings.

The basic flow of timing-constrained mapping is shown in Fig. 10. Steps that have been added from the unconstrained flow, shown in Fig. 4, appear as darkly shaded blocks. These main additions include the following.

- **Delay estimation.** Timing-constrained mapping requires accurate, iterative evaluation of mapped-circuit performance. Our delay estimator uses LUT packing information to compute the arrival time and delay-slack value of each node in the circuit and the largest combinational delay in the circuit. Logic and estimated routing values are used to approximate mapped-circuit performance.
- **Iterative timing-constrained subgraph generation and selection.** Following delay estimation, an iterative mapping process is started to partially transfer design logic to PLAs. During each iteration step, subgraphs suitable for PLA implementation are extracted along noncritical paths. After PLA-feasible subgraphs are identified, the highest

ranking subgraph is packed into an available PLA and circuit delay is updated to account for the delay perturbation due to the included PLA. If fewer than r PLAs have been packed, the next iteration of subgraph generation and selection is started. The iterative search process continues until no additional PLA resources are available.

Each of the steps is integrated into the *hybridmap* flow based on user-preference and clock-period specification.

A. Delay Estimation

The goal of the delay estimator is to compute the design critical path and delay slack values in terms of logic and estimated routing delays. The delay estimation process is composed of delay tracing and delay update phases. The delay tracing procedure [30] computes required signal arrival time and slack values associated with each LUT and any subgraph supernode in a partially mapped design. Each circuit node is assigned a required signal arrival time and slack value equal to that of the associated LUT or supernode. Once an available PLA is packed, the delay updating procedure performs LUT reclustering and design depth value update so that remaining PLAs can be packed.

1) Delay Tracing the Network: The delay tracing procedure uses depth information to identify noncritical paths in the design. For each LUT cluster the required signal arrival time (RT) is computed at each output and input. Subsequently, the slack value (SV) at each cluster output is computed. A detailed example of circuit delay tracing is presented in [30]. Delay tracing has previously been used for LUT-based technology mapping [13] and technology mapping for FPGAs with LUTs and memory blocks [25].

2) Reclustering and Updating the Circuit Delay: Prior to packing an available PLA, the depth of each circuit node is computed in terms of LUTs. Once a PLA is packed, the circuit delay values change along the paths through the PLA, necessitating circuit delay updates. The PLA subgraph is initially collapsed into a multi-input, multi-output supernode. The remaining circuit nodes are subsequently reclustered around the supernodes. Fig. 11 shows the original circuit covered by LUT clusters and a PLA subgraph collapsed to form a supernode.

When updating circuit delay values, LUT clusters are reconstructed with respect to the I/O boundaries of supernodes. The delay updating procedure progresses in two phases. During the first phase, the LUT-clustering procedure described in Section IV-B considers all the nodes in the circuit that are not along the paths through the supernodes. Depth values are computed for the selected nodes and are grouped to form K-LUT clusters. Fig. 11 shows LUT clustering for such selected nodes. During the second phase of the delay updating procedure, the depth along paths through the supernodes are computed using the approach described in Section IV-B. For every supernode sn, the depth at the output (O_{sn}) of sn is computed as the sum of largest depth among inputs that drive O_{sn} and $delay_{\text{PLA}}$, where $delay_{\text{PLA}}$ is the delay of the PLA resource. In Fig. 11, the depth for a selected supernode output, O1, is computed as $\max(d_k \mid k = \{I0, I1, I2, I4\}) + delay_{\text{PLA}} = 1 + 2 = 3$. The depth at O2 is computed as $\max(d_k \mid k = \{I3, I4, I5\}) + delay_{\text{PLA}} = 1 + 2 = 3$. The delay values computed at the outputs of each supernode can subsequently be used to compute the delay of the nodes driven by the supernodes.

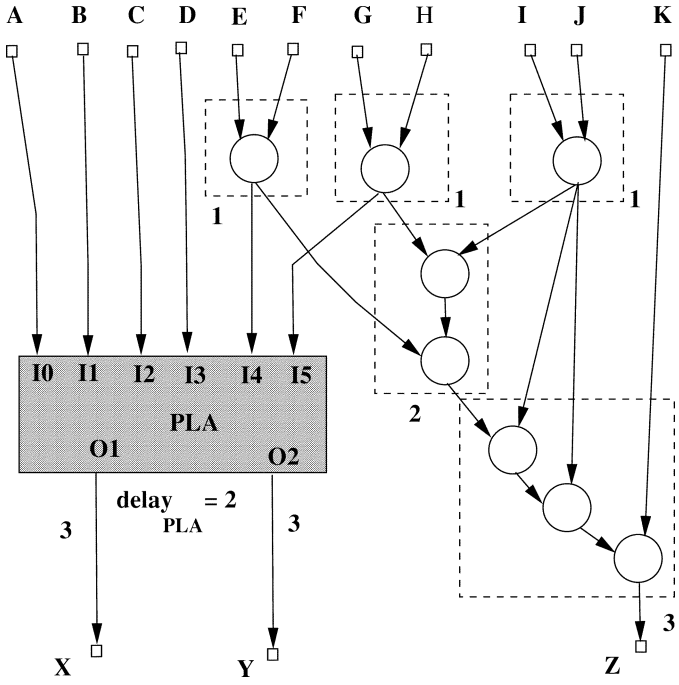


Fig. 11. Example of updating the circuit delay.

B. Timing-Constrained Subgraph Generation

The timing-constrained subgraph generation process is based on the basic subgraph extraction approach described in Section IV-C. The timing-constrained approach searches for subgraphs along noncritical design paths since PLAs inserted along these paths have a higher probability of maintaining LUT-only timing performance. This search does not guarantee LUT-only performance since subsequent routing congestion may lengthen path route lengths, but, as shown in Section VII, in almost all experimental cases, performance is maintained. To control delay, subgraphs are collected only along paths that contain a minimum slack value of $SV_{\min}(= \text{delay}_{\text{PLA}} - \text{delay}_{\text{LUT}})$. In the likely event that a subgraph search encounters a node with slack value $< SV_{\min}$, the subgraph generation process terminates further searches along paths through that node. Thus, during the forward search phase of subgraph extraction, an encountered node u, whose fanout c has a slack value $SV(c)$ ($< SV_{\min}$), is automatically considered a member of the subgraph root set RS. Similarly, during the backward search, any encountered node p whose slack value $SV(p) < SV_{\min}$ is automatically considered an input to the subgraph. These cases are shown in Fig. 12 for a subgraph (S) search starting from node v. If PLA delay ($\text{delay}_{\text{PLA}}$) is four and LUT delay ($delay_{\text{LUT}}$) is one, then, $SV_{\min} = \text{delay}_{\text{PLA}} - \text{delay}_{\text{LUT}} = 3$. As shown in Fig. 12(a), during the forward graph search, the node u is considered a member of $RS(v)$, since $SV(w) = 1$ (< 3). The darkly shaded nodes belong to the $RS(v)$ and the lightly shaded nodes belong to the intermediate nodes of the subgraph. Similarly, as shown in Fig. 12(b), during the backward graph search, the node p is considered as input to S since $SV(p) = 2$ (< 3).

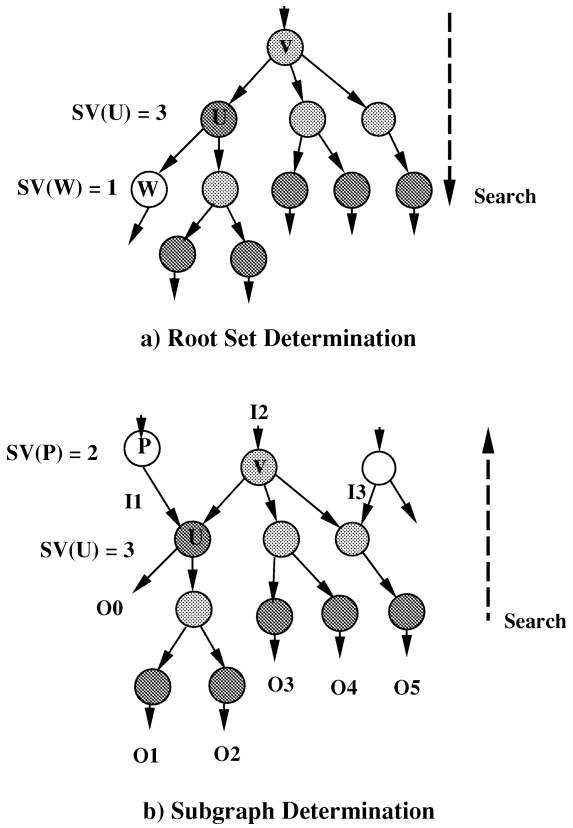


Fig. 12. Root set and subgraph determination – timing-constrained.

In Fig. 12(b), the unshaded nodes and signal $I2$ represent the inputs to the subgraph.

VI. Technology Mapping to the Apex20KE

A. Apex20KE Architecture

To illustrate the benefit of *hybridmap*, our new technology mapping approach was targeted to Altera Apex20KE devices [1]. This hybrid FPGA architecture contains embedded Pterm blocks with 32 inputs, 16 outputs, and 32 Pterms [4]. As shown in Fig. 13, each Pterm block is composed of macrocell structures which can be fed with any combination of the 32 input signals of either polarity. Each macrocell for an Apex20KE Pterm block can either drive a macrocell output or a neighboring macrocell, but not both. As a result, the macrocell architecture does not allow sharing of Pterms/sum of Pterms across multiple outputs. Inputs from neighboring macrocells (parallel expanders) are utilized whenever it is necessary to use more than one macrocell to implement a selected output.

B. Unconstrained Area Minimization

As each vendor PLA architecture differs greatly, some adjustment to our basic mapping algorithms are required to achieve best-possible PLA mapping results. For example, the Apex20KE device allows for the programmable inversion of an AND gate output. The inverter can be programmably set to merge multiple single-input Pterms into one multi-input Pterm using DeMorgan rules (e.g., $a + b + c = \overline{\bar{a}\bar{b}\bar{c}}$). Our Pterm count estimator was modified to consider this architectural feature. The average difference in Pterm reduction predicted by the estimator improved by about 5% when specific Apex architectural features were taken into account.

C. Timing-Constrained Area Minimization

The implementation of logic functions with greater than two product terms requires special processing to meet timing constraints. The delay equation for a function requiring n macrocells is $d_{\text{out}} = 2 \times d_{\text{gate}} + (n-1) \times d_{\text{pexp}}$, where d_{gate} and d_{pexp} are the delays of a macrocell and parallel expander, respectively. As $d_{\text{gate}} > 3 \times d_{\text{LUT}}$ for the Apex20KE, the use of parallel expanders results in a significant delay for wide Pterm functions. For functions requiring more than four Pterms, Pterm outputs can be combined using four-LUTs. This approach also allows for function complement generation that can be used during minimization. The added LUTs lead to a minimal reduction in LUT coverage.

VII. Results

To evaluate the performance of *hybridmap*, the MCNC benchmarks [8] listed in Table II were mapped to hybrid FPGA architectures. Unless specified otherwise, the target hybrid architecture is the Altera Apex20KE [1]. Results were obtained on a 386-MHz Celeron-based PC containing 128-MB RAM. All experiments, unless noted, involved the use of the Pterm count estimator.

The first experiment conducted with our system was to evaluate the amount of LUTs that could be absorbed into PLAs for a typical design without timing constraints (unconstrained). Two initial representations of input circuits were considered, designs reduced to two-input gates and designs preclustered to four-input nodes (LUTs). Previous embedded memory mapping approaches [26] have used four-LUTs as the basis for subgraph search. For our system, both two-input gate representations and four-bounded representations are supported. For the benchmarks listed in Table II, two separate input graph representations were constructed. For the two-bounded case, each input netlist was optimized with SIS (script.algebraic) and *dmig* [15] to create two-bounded nodes. For the four-bounded (four-LUT) case, each netlist was optimized with SIS and Flowmap [15]. In both cases, resulting circuits were processed by *hybridmap*. Results in Table II indicate remaining LUT counts after *hybridmap* completion for a range of PLAs per device and average LUT coverage per PLA. Post-hybridmap LUT processing was performed by Flowmap. It can be seen that two-bounded graphs allow for greater or equal search flexibility. For 10 PLAs per device, approximately 17 LUTs could be covered per PLA. Note that LUT coverage per PLA decreased as the number of PLAs per device increased. Area-driven results with PLA counts similar to the Apex device ($r = 5$) indicate a 15% reduction in LUT usage.

A. Pterm Count Estimation

In a second experiment, the benefit of the Pterm-count estimator was evaluated. Table III indicates leftover LUT counts for *hybridmap* run both with and without Pterm estimation for

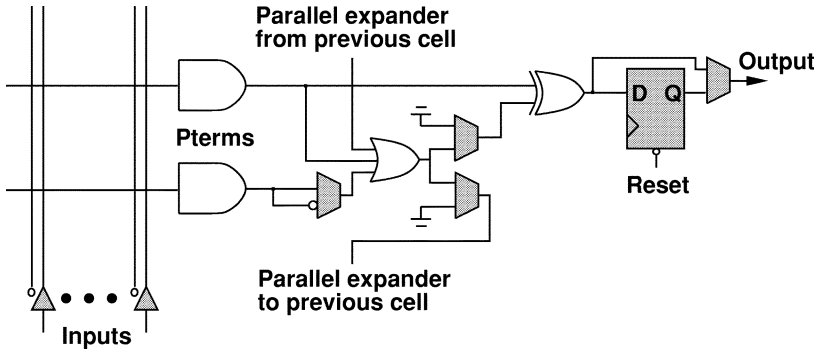


Fig. 13. Apex20KE macrocell.

TABLE II
UNCONSTRAINED MAPPING OF DESIGNS TO r PLAS. RESULTS WERE GENERATED WITH THE USE OF THE PTERM COUNT ESTIMATOR

Ckt. Name	Orig. 4-LUTs	4-LUT based Input Graph Final 4-LUT Count r = 1	r = 5	r = 10	2 bounded Input Graph Final 4-LUT Count r = 1	r = 5	r = 10	LUTs covered per PLA r = 1	r = 5	r = 10
C5315	608	585	559	504	570	529	470	38.0	15.8	13.8
C7552	712	700	668	635	701	657	621	11.0	11.0	9.1
apex6	393	362	258	141	364	271	137	29.0	24.4	15.6
dalu	426	413	362	312	414	370	333	12.0	11.2	9.3
des	1583	1547	1441	1275	1553	1420	1318	30.0	32.6	26.5
duke2	240	211	124	34	217	133	47	23.0	21.4	19.3
e64	270	245	163	78	229	155	62	41.0	23.0	20.8
ex5p	1301	1277	1201	1119	1275	1216	1142	26.0	17.0	15.9
i10	978	958	888	807	919	838	804	59.0	28.0	17.4
pair	604	583	519	455	581	511	417	23.0	18.6	18.7
rot	304	280	226	164	279	198	145	25.0	21.2	15.9
s1196	260	226	141	47	237	122	50	23.0	27.6	21.0
s298	1260	1236	1134	1010	1236	1152	1036	24.0	21.6	22.4
Total/Ave.	8939	8623	7684	6581	8575	7572	6582	28.0	21.0	17.4

TABLE III
UNCONSTRAINED *HYBRIDMAP* RESULTS WITH AND WITHOUT PTERM COUNT ESTIMATOR

Ckt. Name	Orig. 4-LUTs	Without Pterm Estimation Final 4-LUTs [Tot. Run-Time min:sec] r = 1	r = 5	r = 10	With Pterm Estimation Final 4-LUTs [Tot. Run-Time min:sec] r = 1	r = 5	r = 10
C5315	608	585 [7:38]	559 [7:44]	504 [7:49]	570 [6:43]	529 [6:50]	470 [6:56]
C7552	712	701 [1:41]	657 [1:47]	642 [1:52]	701 [1:59]	657 [2:15]	621 [2:27]
apex6	393	368 [0:11]	286 [0:14]	201 [0:16]	364 [0:22]	271 [0:41]	137 [0:54]
dalu	426	414 [4:13]	370 [4:16]	333 [4:18]	414 [7:20]	370 [7:34]	333 [7:56]
des	1583	1552 [12:20]	1427 [12:26]	1361 [12:44]	1553 [14:09]	1420 [14:46]	1318 [14:54]
duke2	240	214 [0:09]	164 [0:10]	81 [0:11]	217 [0:04]	133 [0:07]	47 [0:11]
e64	270	229 [0:05]	155 [0:06]	62 [0:07]	229 [0:05]	155 [0:06]	62 [0:07]
ex5p	1301	1289 [8:48]	1254 [8:53]	1219 [8:56]	1275 [4:21]	1216 [4:23]	1142 [4:28]
i10	978	934 [8:08]	868 [8:25]	807 [8:45]	919 [7:32]	838 [7:43]	804 [7:59]
pair	604	587 [1:28]	518 [1:35]	430 [1:44]	581 [2:23]	511 [2:47]	417 [3:40]
rot	304	279 [0:19]	217 [0:22]	148 [0:25]	279 [0:43]	198 [1:01]	145 [1:28]
s1196	260	233 [0:32]	159 [0:42]	92 [0:48]	237 [0:45]	122 [2:04]	50 [2:32]
s298	1260	1235 [4:29]	1157 [4:39]	987 [4:56]	1236 [5:18]	1152 [5:29]	1036 [5:58]
Total	8939	8620 [46:01]	7791 [47:19]	6867 [48:51]	8575 [45:44]	7572 [50:56]	6582 [54:30]

a range of PLAs per device (r). For runs without Pterm estimation, all subgraphs that exceeded 32 Pterms were immediately eliminated from consideration without regard to their likely *post-minimization* Pterm count. Total *hybridmap* run time is indicated in brackets. *Post-hybridmap* LUT processing was performed with Flowmap. As seen in Table III, the Pterm estimator allows for approximately 4% improved overall LUT coverage for a modest increase in overall *hybridmap* run time. In some cases, the use of the Pterm estimator reduced run time since subgraphs were more aggressively extracted, reducing the search space. All designs were processed starting from a two-input gate representation.

B. Comparisons to Previous Work

As discussed in Section II-C, several previous approaches to hybrid technology mapping with no timing constraints have been developed. In a customized experiment, *hybridmap* was

TABLE IV
AREA COMPARISON WITH NODE-BASED HYBRID MAPPING [31]

Ckt	LUTs-only		Node-based [31]			Hybridmap		
Name	4-LUTs	Depth	PLAs	4-LUTs	Depth	PLAs	4-LUTs	Depth
s1423	154	21	19	72	17	19	95	15
frg2	324	8	30	123	6	30	107	6
x3	282	7	25	98	4	25	109	5
dalu	357	13	27	106	10	27	104	9
sbc	266	8	26	105	8	26	124	5
cps	520	8	53	209	10	53	221	8
s1488	219	7	21	81	6	21	85	6
scf	300	6	33	126	6	33	116	7
apex2	905	16	90	366	13	90	355	9
alu4	666	10	69	286	10	69	245	10
Total	3993		393	1572		393	1561	

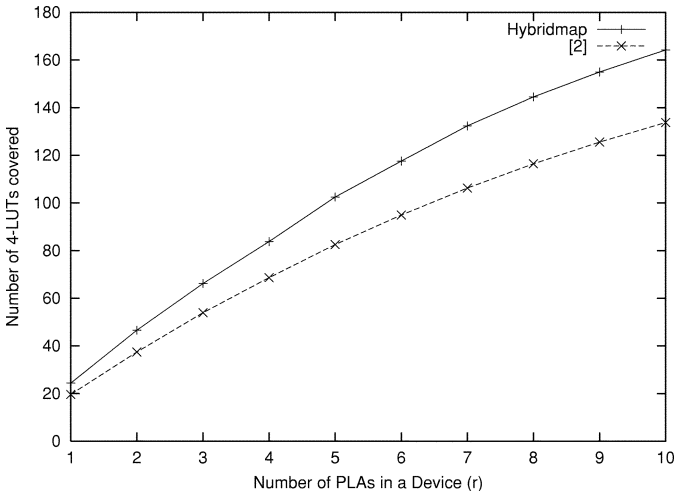


Fig. 14. Hybridmap comparison to Lin and Wilton [2].

compared to results reported in [31] for hybrid devices containing low fanout PLAs ($i_m = 16$ inputs, $o_m = 3$ outputs, and $p_m = 10$ Pterms). As shown in Table IV, the results obtained from *hybridmap* when targeting these small PLAs were competitive with previously-reported work. The LUT counts noted in the fifth and eighth column indicate remaining LUTs after logic has been mapped to PLAs. The optimized benchmarks used for these experiments were previously used in [31] and were obtained from Kaviani. Note that the number of PLAs per device varied from design to design.

Fig. 14 compares *hybridmap* results against those presented by Lin and Wilton in [2] (Fig. 12, row 1). The target architecture for both cases is the Apex20KE architecture with PLA counts ranging from 1 to 10. The input to the PLA mapping approach described in [2] is a four-LUT mapped circuit from which PLA subgraphs are extracted based on a search algorithm. The uncovered nodes in the design at the end of the PLA mapping process are counted to obtain the final LUT count. The benchmark circuits used by Lin and Wilton were obtained from Lin and were input to *hybridmap*. In order to maintain a common ground for comparison, no preprocessing operations such as logic optimization or two-input gate decomposition were performed on the input benchmark circuits. The post-mapping four-LUT count of each benchmark circuit was obtained by counting the number of nodes in the four-LUT partition output of *hybridmap*. It can be seen that, as the number of available PLAs increases, *hybridmap* achieves better LUT coverage than the approach presented in [2]. A maximum improvement of 22% was achieved for 10 PLAs per device ($r = 10$). The improved PLA packing density obtained by *hybridmap* is attributed mainly to two of the procedures available in *hybridmap* but not in [2]: subgraph-based logic extraction accentuated by hill climbing and Pterm estimation targeting Apex20KE devices.

Since the work described in this paper is the first reported timing-constrained mapping approach for hybrid FPGAs, it was not possible to compare against previous work in this area.

C. Mapping to Altera Apex20KE Devices

As a final experiment, *hybridmap* was applied in unconstrained and timing-constrained mode to the benchmark circuits listed in Table V. For delay and area comparison purposes, designs were initially mapped entirely to LUTs using Altera Quartus v2000.02 [5], the commercially available Altera tool set. The critical path of each mapped circuit was then used as the minimum clock frequency constraint for *hybridmap*. Following this initial mapping, *hybridmap* was then applied to the initial (non-Quartus mapped) circuits and Pterm and LUT partitions were created. These partitions were subsequently mapped to EP20KE device resources using Quartus with *speed* as the synthesis objective. For each design, the smallest EP20KE device which could support LUTs-only mapping *and* design pin constraints was targeted.

Table V compares the area and delay values obtained for four-LUT implementation against that obtained for *hybridmap* timing-constrained and unconstrained implementation. LUT values indicate the number of LUTs remaining after *hybridmap* processing. Note that not all designs have logic mapped to Pterms in the timing-constrained mode. Logic could not be migrated to Pterms for designs with a zero in column 6 without increasing the circuit critical path. The bottom row of the table presents the arithmetic sum of the results obtained for benchmark circuits. *Hybridmap* meets the delay value incurred by a purely four-LUT implementation, and packs about 8% of initial design LUTs to Pterms. For the unconstrained case, all designs had some logic mapped to Pterms. Overall, 14% of logic could be mapped from LUTs to Pterms. It was estimated

TABLE V
LUT COVERAGE FOR APEX20KE DEVICES (SPEED GRADE -1)

Ckt Name	Target Device	No. Pins	Timing Constrained					Unconstrained	
			LUTs-only		Hybrid			Hybrid	
			Area	Delay	Area		Delay	Area	
			LUTs	nS	PLAs	LUTs covered	nS	PLAs	LUTs covered
C5315	$EP20K200$	484	525	20.3	0	0	20.3	5	31
C7752	$EP20K200$	484	521	28.9	5	37	27.2	5	47
apex6	$EP20K200$	484	388	14.5	5	121	14.5	5	121
dalu	$EP20K60$	208	568	26.2	5	175	24.1	5	175
des	$EP20K200$	672	1555	29.4	0	0	29.4	5	93
duke2	$EP20K60$	144	222	14.1	0	0	14.1	5	78
e64	$EP20K60$	208	258	15.1	0	0	15.1	5	93
ex5p	$EP20K60$	144	1251	20.9	0	0	20.9	5	61
i10	$EP20K200$	652	930	38.7	5	38	37.5	5	57
pair	$EP20K200$	484	769	19.5	5	111	19.2	5	111
rot	$EP20K100$	324	271	18.9	5	53	18.1	5	84
s1196	$EP20K60$	144	263	17.1	0	0	17.1	5	122
s298	$EP20K60$	144	1157	33.8	5	152	33.6	5	152
Total/Ave.			**8678**	22.9	35	**687**	22.4	65	**1225**

with *hybridmap* that if Pterm blocks required 2X, rather than 3X, the delay of LUTs, timing-constrained LUT coverage would rise from 8% to 11.8%, approaching 14% unconstrained coverage.

VIII. CONCLUSION

Hybrid devices facilitate efficient area and delay tradeoffs for FPGA designs. In this paper, heuristic techniques to automatically identify design partitions for implementation in PLA-based logic resources have been described. A subgraph extraction approach based on heuristic search and hill-climbing was found to quickly identify feasible PLA subgraphs including those with reconvergent paths. A Pterm-count estimator has been developed which is sufficiently fast enough to be used in the inner loop of the subgraph generation. An area estimator further guides the subgraph selection process by estimating the LUT count coverage due to each subgraph. *Hybridmap* has been developed to support both unconstrained and timing-constrained area optimization. The technology mapping tool, evaluated using Altera's Apex20KE devices [1], reveals that, on average, 8% of four-LUT area can be transferred to Pterms while preserving device timing performance and 14% can be transferred without timing constraints. This provides additional space for subsequent design additions or for migration to a smaller FPGA device.

ACKNOWLEDGMENT

The authors are thankful to A. Kaviani for discussions regarding the hybrid technology mapping problem and for providing the circuits that were previously used in [31]. Thanks are due to E. Lin for providing the circuits used in [2]. They are grateful to A. Venkataraman, A. Laffely, and J. Liang for their valuable feedback regarding this work.

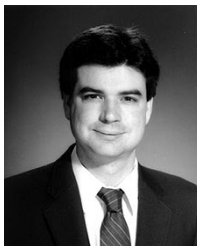
REFERENCES

[1] *Apex20KE Data Sheet*, Altera Corporation, San Jose, CA, 1999.
[2] E. Lin and S. Wilton, "Macrocell architectures for product term embedded memory arrays," *Field Program. Logic Applicat.*, pp. 48–58, Aug. 2001.
[3] A. Kaviani and S. Brown, "The hybrid field-programmable architecture," *IEEE Design Test Comput.*, vol. 16, pp. 74–83, Apr. 1999.
[4] F. Heile and A. Leaver, "Hybrid product term and LUT based architectures using embedded memory blocks," in *ACM 7th Int. Symp. Field-Program. Gate Arrays*, Feb. 1999, pp. 13–16.
[5] *Quartus User Guide*, Altera Corporation, San Jose, CA, 2001.
[6] *Synplify User Guide*, Synplicity, Inc, Sunnyvale, CA, 2001.
[7] R. Brayton, A. Sangiovanni-Vincentelli, G. Hachtel, and C. McMullin, *Logic Minimization Algorithms for Digital Circuits*. Norwell, MA: Kluwer, 1984.
[8] S. Yang, "Logic Synthesis and Optimization Benchmarks User Guide, vers. 3.0," Microelectronics Center of North Carolina, Research Triangle Park, NC, 1991.
[9] A. Farrahi and M. Sarrafzadeh, "Complexity of the lookup-table minimization problem for FPGA technology mapping," *IEEE Trans. Computer-Aided Design*, vol. 13, pp. 1319–1332, Nov. 1994.
[10] J. Cong and S. Xu, "Technology mapping for FPGA's with embedded memory blocks," in *Proc. ACM/SIGDA 6th Int. Symp. Field-Programmable Gate Arrays*, Feb. 1998, pp. 179–188.
[11] J. Cong and Y. Ding, "Tutorial and survey paper – Combinational logic synthesis for LUT based field programmable gate arrays," *ACM Trans. Design Automation Electron. Syst.*, vol. 1, no. 2, pp. 145–204, 1996.
[12] R. J. Francis, J. Rose, and K. Chung, "Chortle: A technology mapping program for lookup table-based field programmable gate arrays," in *Proc. 27th ACM/IEEE Design Automation Conf.*, June 1990, pp. 613–619.
[13] J. Cong and Y. Ding, "On area/depth trade-off in LUT-based FPGA technology mapping," in *Proc. ACM/IEEE 30th Design Automation Conf.*, June 1993, pp. 213–218.
[14] 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.*, vol. 9, pp. 7–20, Sept. 1992.
[15] 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.
[16] J. Cong, Y. Ding, A. Kahng, P. Trajmar, and K. C. Chen, "An improved graph-based FPGA technology mapping for delay optimization," in *Proc. IEEE Int. Conf. Computer-Aided Design*, Oct. 1992, pp. 154–158.
[17] J. Cong and Y. Ding, "An optimal performance-driven technology mapping algorithm for LUT-based FPGA's under arbitrary net-delay models," in *Proc. Int. Conf. Computer-Aided Design*, Aug. 1993, pp. 599–604.
[18] H. Yang and D. F. Wong, "Edge-map: Optimal performance driven technology mapping algorithm for iterative LUT-based FPGA designs," in *Proc. IEEE Int. Conf. Computer-Aided Design*, Nov. 1994, pp. 150–155.
[19] J. Cong and S. Xu, "Performance-driven technology mapping for heterogeneous FPGA's," *IEEE Trans. Computer-Aided Design*, vol. 19, pp. 1268–1281, Nov. 2000.

[20] J. He and J. Rose, "Technology mapping for heterogeneous FPGA's," in ACM Int. Symp. Field-Programmable Gate Arrays, Feb. 1994.
[21] J. Cong, H. Huang, and X. Yuan, "Technology mapping for k/m-macrocell based FPGA's," in *Proc. ACM/SIGDA 8th Int. Symp. Field-Program. Gate Arrays*, Feb. 2000, pp. 51–59.
[22] D. Chen, J. Cong, M. Ercegovac, and Z. Huang, "Performance-driven mapping for CPLD architecture," in *Proc. ACM/SIGDA 9th Int. Symp. Field-Program. Gate Arrays*, Feb. 2001, pp. 39–47.
[23] *Flex10K Data Sheet*, Altera Corporation, San Jose, CA, 2001.
[24] *Virtex Data Sheet*, Xilinx Corporation, San Jose, CA, 2001.
[25] J. Cong and K. Yan, "Synthesis for FPGA's with embedded memory blocks," in *Proc. ACM/SIGDA 8th Int. Symp. Field-Program. Gate Arrays*, Feb. 2000, pp. 75–82.
[26] S. Wilton, "SMAP: Heterogeneous technology mapping for area reduction in FPGA's with embedded memory arrays," in *Proc. ACM 6th Int. Symp. Field-Program. Gate Arrays*, Feb. 1998, pp. 171–178.
[27] A. Kaviani, "Novel Architectures and Synthesis Methods for High Capacity Field-Programmable Gate Arrays," Ph.D. dissertation, Dept. of Elec. Comput. Eng., Univ. of Toronto, Toronto, ON, Canada, 1999.
[28] S. Krishnamoorthy, S. Swaminathan, and R. Tessier, "Area optimized technology mapping for hybrid FPGA's," *Field Program. Logic Applicat.*, pp. 180–191, Aug. 2000.
[29] E. M. Sentovich, K. J. Singh, L. Lavagno, C. Moon, R. Murgai, A. Saldanha, H. Savoj, P. Stephan, R. Brayton, and A. Sangiovanni-Vincentelli, "SIS: A System for Sequential Circuit Analysis," Electron. Res. Lab., Univ. of California, Berkeley, CA, Memorandum no. UCB/ERL M92/41, 1992.
[30] S. Devadas, A. Ghosh, and K. Keutzer, *Logic Synthesis*. New York: McGraw–Hill, 1994.
[31] A. Kaviani and S. Brown, "Technology mapping issues for an FPGA with lookup tables and PLA-like blocks," in *Proc. ACM/SIGDA 8th Int. Symp. Field-Program. Gate Arrays*, Feb. 2000, pp. 60–66.

Srini Krishnamoorthy (S'00) received the B.E. degree in electrical and electronics engineering from the A.C. College of Engineering and Technology, Tamilnadu, India, in 1993, and the M.Tech. degree in communications engineering from the Indian Institute of Technology, Bombay, India, in 1997. He is currently pursuing the Ph.D. degree in electrical and computer engineering at the University of Massachusetts, Amherst.

His research interests include FPGAs, logic synthesis, graph algorithms, combinatorial optimization, and CAD techniques for very large scale integration.



Russell Tessier (M'00) received the B.S. degree in computer engineering from Rensselaer Polytechnic Institute, Troy, NY, in 1989, and the S.M. and Ph.D. degrees in electrical engineering from the Massachusetts Institute of Technology, Cambridge, in 1992 and 1999, respectively.

He is an Assistant Professor of electrical and computer engineering at the University of Massachusetts, Amherst, where he leads the Reconfigurable Computing Group. He is a founder of Virtual Machine Works, Cambridge, MA, a logic emulation company, and has also been with BBN and Ikos Systems. His research interests include computer architecture, FPGAs, and system verification.