Technology Mapping and Architecture Evalution for *k/m*-Macrocell-Based FPGAs

JASON CONG and HUI HUANG
University of California, Los Angeles
and
XIN YUAN
IBM Corporation

In this article, we study the technology mapping problem for a novel field-programmable gate array (FPGA) architecture that is based on k-input single-output programmable logic array- (PLA-) like cells, or, k/m-macrocells. Each cell in this architecture can implement a single output function of up to k inputs and up to m product terms. We develop a very efficient technology mapping algorithm, k_m_flow, for this new type of architecture. The experimental results show that our algorithm can achieve depth-optimality on almost all the testcases in a set of 16 Microelectronics Center of North Carolina (MCNC) benchmarks. Furthermore it is shown that on this set of benchmarks, with only a relatively small number of product terms ($m \leq k+3$), the k/m-macrocell-based FPGAs can achieve the same or similar mapping depth compared with the traditional k-input single-output lookup table- (k-LUT-) based FPGAs. We also investigate the total area and delay of k/m-macrocell-based FPGAs and compare them with those of the commonly used 4-LUT-based FPGAs. The experimental results show that k/m-macrocell-based FPGAs can outperform 4-LUT-based FPGAs in terms of both delay and area after placement and routing by VPR on this set of benchmarks.

Categories and Subject Descriptors: B.6.3 [**Logic Design**]: Design Aids; B.7.1 [**Integrated Circuits**]: Types and Design Styles—*Gate arrays*

General Terms: Algorithms, Design

Additional Key Words and Phrases: PLD, FPGA, CPLD, technology mapping

1. INTRODUCTION

The programmable logic devices (PLDs) have been widely used to implement small to medium sized digital circuits. There are two major types of

This work was partially supported by Altera Corp. and Vantis Corp. under the California MICRO program.

X. Yuan was affiliated with the University of California, Los Angeles, Los Angeles, CA, at the time of the research for this article.
Authors' addresses: J. Cong, Department of Computer Science, University of California, Los Angeles, Los Angeles, CA 90095; email: cong@cs.ucla.edu; X. Yuan, System and Technology Group, IBM Corporation, 1000 River St., Mail drop 862F, Essex Junction, VT 05452; email: xinyuan@us.ibm.com.

PLDs—field programmable gate arrays (FPGAs), which usually consist of small programmable logic cells, such as k-input single-output lookup tables (k-LUTs), and complex programmable logic devices (CPLDs), which are based on multiple-input and multiple-output programmable logic array- (PLA-) like logic cells. Both FPGAs and CPLDs have been widely used.

Most commonly used FPGAs are based on k-LUTs. Every k-LUT can implement any function with no more than k inputs. In practice, k is usually small, as the area of a k-LUT grows exponentially with large k. For example, LUTs of four to six inputs are widely used in commercial FPGAs [Altera Corp. 2001; Xilinx Inc. 2001]. On the other hand, CPLDs usually have large basic cells. Each PLA cell can have a large number of inputs (typically between 30 and 40), product terms (typically between 50 to 100), and multiple outputs (16, for example) [Altera Corp. 2000; Cypress Semiconductor Corp. 2000; Lattice Semiconductor Corp. 2000]. As a result, a single PLA cell is able to implement multiple functions with wide inputs.

Rose et al. [1990] showed that among k-LUT cells, the four-input single-output LUT cell yields the smallest FPGA area for a wide range of programming technologies and routing pitches. Most commercially available FPGAs indeed use LUTs with an input size of four or five. Kouloheris and El Gamal [1992] investigated the best granularity for PLA-based CPLDs and found that the total CPLD area is smallest if each basic cell has 8 to 10 inputs, 3 to 4 outputs, and 12 to 13 product terms. The number of product terms is restricted to grow linearly as input size increases [Kouloheris 1993]. In practice, however, most commercially available CPLDs use much larger PLA-like logic cells. Since FPGAs use small programmable cells, they can often offer higher density and capacity, at a price of possibly larger and somewhat unpredictable delay, as the critical path often needs to go through multiple levels of programmable cells connected by the programmable interconnect. On the other hand, CPLDs are usually faster as the programmable cells are much larger, which results in fewer levels of the logic. (The worst-case delay in CPLD also tends to be more predictable as the level of the logic is usually determined by the architecture and can be estimated by the designer.) However, CPLDs usually offer considerably lower logic density. We believe that this is due to two reasons: (a) it is inherently difficult to map logic into multioutput PLA-like programmable cells, as most technology mapping techniques have been developed for single-output logic cells; and (b) the difficulty associated with synthesis/mapping for PLA-based CPLD devices in turn resulted in very limited studies on this topic—the only related works we can find were the DDMap [Kouloheris 1993], a fast heuristic partition method for PLA-based architecture proposed in Hasan et al. [1992], TEMPLA [Anderson and Brown 1998], and PLAmap [Cong et al. 2001]. In the later 1990s, with the introduction of hybrid FPGA families [Kaviani and Brown 1999], a few of mapping algorithms for a hybrid architecture of LUTs and PLAs have been reported [Kaviani 1999; Krishnamoorthy and Tessier 2003]. (In comparison, there have been much more extensive studies on LUT-based FPGAs. A comprehensive survey up to 1997 was reported in Cong and Ding [1996].)

The need to reduce the logic levels (and associated interconnects!) to improve circuit performance, the intention to avoid the mapping problem for multioutput functions, and the hope to leverage a large amount of research results on synthesis and mapping for LUT-based FPGAs, seem to suggest that we should consider FPGAs with LUTs of much larger numbers of inputs. However, the area of a k-LUT grows expontentially with respect to k. Using k-LUTs with large k may considerably lower chip density. Therefore, we have to explore other alternatives. It has been reported that the functions mapped into large LUTs usually use considerably fewer product terms than the capacity of the lookup table [Kouloheris and Gamal 1992; Kouloheris 1993]. For instance, the utilization of a lookup table cell with K-inputs and N-outputs, which is defined as the ratio of the number of product terms used per cell to the cell capacity, $N \cdot 2^{K-1}$, is less than 0.1 for $K = 7, N = 1$. This leads us to consider an FPGA architecture based on k-input single-output PLA-like logic cells. Each cell can implement a single output function of up to m product terms and up to k inputs. Such a cell is called a *k/m-macrocell* throughout this article. A k/m-macrocell differs from a k-LUT in that each macrocell can implement only a subset of all possible k-input functions. A k/m-macrocell is different from a general PLA-like block used in most CPLDs, as each k/m-macrocell has a single output. If we choose m to be small, k/m-macrocells are much smaller than k-LUTs. Therefore, it is possible to use k/m-macrocells with larger input size in order to get a smaller logic depth and less interconnect without considerably lowering the logic density.

In this article, we develop a very efficient technology mapping algorithm, named *k_m_flow*, for this new type of architecture. The experimental results show that our algorithm can achieve depth-optimality on almost all the testcases in a set of 16 Microelectronics Center of North Carolina (MCNC) benchmarks. Furthermore it is shown that on this set of benchmarks, with only a relatively small number of product terms ($m \le k+3$), the k/m-macrocell-based FPGAs can achieve the same or a similar mapping depth compared with the traditional k-LUT based FPGAs. We also investigate the total area and delay of k/m-macrocell-based FPGAs and compare them with those of the commonly used 4-LUT-based FPGAs. The experimental results show that k/m-macrocell-based FPGAs can outperform 4-LUT-based FPGAs in terms of both delay and area after placement and routing by versatile place route (VPR) on this set of benchmarks.

The rest of this article is organized as follows. Section 2 formulates the problem. Section 3 introduces a technology mapping algorithm for k/m-macrocell-based FPGAs. Section 4 further investigates the area and delay of k/m-macrocell-based architecture. We draw our conclusions based on experimental results in Section 5. An extended abstract of this work was presented at the FPGA Symposium in 2000 [Cong et al. 2000].

2. DEFINITIONS AND PROBLEM FORMULATION

We first review some terminologies defined in Cong and Ding [1994]. A Boolean network can be represented as a directed acyclic graph (DAG) where 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. We use *input*(v) to denote the set of nodes which are fanins of gate v. We assume the network is k-bounded, that is, for each node v in the network, $|input(v)| \leq k$.[1] A cone at node v, denoted as C_v, is a subgraph consisting of v and its predecessors such that any path connecting a node in C_v and v lies entirely in C_v. The notation of $input(C_v)$ is also used to represent the set of distinct nodes outside C_v which supply inputs to the gates in C_v. A maximum cone at v, also the fanin network of v, denoted as N_v, is a cone consisting of v and all of its predecessors. The *level* of a node v is the length of the longest path from any PI node to v. The level of a PI is zero. The *depth* of a network is the largest node level in the network.

A cone C_v is said to be *k-feasible* if and only if $|input(C_v)| \leq k$. Similarly, C_v is said to be *m-packable* if and only if its function has a sum-of-product representation with no more than m product terms. C_v is said to be *k/m-feasible* if it is both *k-feasible* and *m-packable*. Please note that the word *feasible* usually refers to the number of inputs of a macrocell, and *packable* refers to the number of product terms. The only exception is *k/m-feasible*, which is a shortened version of *k-feasible and m-packable*. It is obvious that a *k/m-feasible* cone can be implemented by a k/m-macrocell. A network is called *m-packable* if the function of any node in the network has a sum-of-product representation with no more than m product terms.

Several concepts about cuts in a network will be used in our discussion. Given a network N with a source s and a sink t, a cut (X, X') is a partition of the nodes in the network such that $s \in X, t \in X'$, and no nodes in X' provide input to any node in X. Clearly X' may be considered a cone at t inside network N. Therefore we can apply the previous definitions on k/m-feasibility to cuts. A cut (X, X') is said to be *k-feasible* if and only if X' is a k-feasible cone. The cut is said to be *m-packable* if and only if X' is an m-packable cone. A *k/m-feasible* cut is one that is both k-feasible and m-packable. For every node v and its fanin network N_v, a cut (X, X') in N_v is a partition of the nodes such that all the PI nodes belong to X and v belong to X'. It is clear that every cone rooted at v corresponds to a cut in N_v.

The technology mapping problem for k/m-macrocell-based FPGAs is to cover a given k-bounded m-packable Boolean network with a set of k/m-feasible cones. Note that we allow these cones to overlap, that is, nodes may be duplicated in the mapping solutions. This problem is NP-Hard as the two level minimization problem is NP-Hard.

Throughout the discussion on the technology mapping algorithm (Section 3), unit delay and unit area models are used. That is, variation of interconnection delay and routing area is not directly considered during technology mapping of the original network. Each k/m-macrocell contributes a constant delay independent of the function it implements. Each cell is counted as a unit when we evaluate the area; hence the total area of the mapping solution equals to

[1]Any network can be fully decomposed into a two-bounded network without deteriorating the mapping quality [Cong and Hwang 1996].

the total number of macrocells. Such simplification is reasonable because the layout information is not available yet. For architecture evaluation in Section 4, however, we will use more accurate delay and area models with consideration of the interconnect. We use a well-known FPGA placement and routing tool (VPR [Betz et al. 1999]) to get the total area and critical path delay after layout for comparison. To avoid confusion, we use *depth* and *number of macrocells* in Section 3 to refer to the delay and area under the unit delay and the unit area model.

3. TECHNOLOGY MAPPING FOR *k/m*-MACROCELLS

3.1 Overview

A k/m-macrocell can be regarded as a k-LUT with an additional restriction that it can only implement logic functions with no more than m product terms. Therefore, it is natural to start with the k-LUT mapping problem since it has been intensively studied in the past decade.

Currently, there are three major approaches to LUT-based FPGA mapping: tree-based mapping (e.g., Chortle-crf [Francis et al. 1991a], Chortle-d [Francis et al. 1991b]), flow-based mapping (e.g., FlowMap [Cong and Ding 1994]), and cut-enumeration-based mapping (e.g., PREATOR [Cong et al. 1999]). A more comprehensive survey is available in Cong and Ding [1996]. Tree-based mapping algorithms partition the network into trees and handle each tree separately. Each individual tree can be mapped optimally but a prior tree partitioning often compromises the mapping quality. Usually they are fast heuristic algorithms. Flow-based mapping algorithms are based on the theorem of max-flow-min-cut and the computation of network flow. It can generate a depth optimal mapping solution in polynomial time. However, flow-based algorithms lack flexibility as they find only one or two depth-optimal min-cuts for every node. On the other hand, cut-enumeration-based approaches can find out many, if not all, possible cuts for every node. They offer high flexibility and can achieve optimality with more constraints, but they are considerably slower than tree-based or flow-based methods.

The approach we present here, called *k_m_flow*, is a hybrid of flow-based and cut enumeration-based methods. We try to find a k/m-feasible cut for every node first by flow computation. If that fails, we turn to cut enumeration.

The k_m_flow algorithm consists of two phases—(i) labeling the network and (ii) mapping the network into macrocells. The labeling phase is trying to find a k/m-feasible cut for each node for depth minimization. The mapping phase generates k/m-macrocells in the mapping solution according to the labels and cuts computed in the labeling phase.

3.2 Nonmonotone Properties

For every node v, let N_v be the fanin network consisting of node v and all its predecessors. We assume that there is a given label $label(v)$ associated with each node v. We also define $label^*(v)$, the optimal mapping depth of v, to be the minimum depth of the k/m-macrocell mapping solution for N_v. The labeling phase for k/m-macrocell mapping is similar to that in the FlowMap algorithm [Cong and Ding 1994]. It finds a k/m-feasible cut for every node v and computes a label for v to minimize the depth of the k/m-macrocell implementing node v in the mapping solution. Ideally, we would like the computed label to be equal to the optimal mapping depth, that is, $label(v) = label^*(v)$ for every node v in the network, as is in the case with the FlowMap algorithm for k-LUT mapping. However, it is more difficult to do so for the k/m-macrocell-based mapping due to the nonmonotone property of the clustering constraints and the non-monotone property of the optimal depths.

3.2.1 *Nonmonotone Property of Clustering Constraints.* The fundamental difficulty of k/m-macrocell-based FPGA mapping is that the constraints on the number of inputs and the number of product terms of a k/m-macrocell are not monotone clustering constraints. That is, if a cone C_v is k-infeasible (or m-unpackable), it does not guarantee that all its supercones (i.e., those cones including C_v) are k-infeasible (or m-unpackable). As a result, a k-infeasible (or m-unpackable) cone C_v could become k-feasible (or m-packable) by including more nodes into it. Note that the LUT mapping problem also has this non-monotone property.

3.2.2 *Nonmonotone Property of Optimal Mapping Depths.* In addition, the optimal k/m-macrocell mapping depth is not monotone either. The optimal mapping depth is monotone if $label^*(v) \geq label^*(u)$ as long as u is an input to v.

An example of the nonmonotone properties is shown in Figure 1. It is a portion of a two-bounded network and we do not show the complete network (for example, output f^1, f^2 can be drivers to other gates that are not shown in Figure 1). Assuming $k = 4$ and $m = 4$, cone C_{f1} is not *4-packable* while a larger cone C_f is both *4-feasible* and *4-packable*. The optimal depth to implement C_{f1} is 2 with 3 4/4-macrocells (as shown in the shaded regions). However, C_f can be implemented with only 1 4/4-macrocell and therefore the optimal mapping depth for C_f is 1, that is, for node f and its input node f_1, $label^*(f) = 1 < 2 = label^*(f_1)$. Note that for the LUT mapping problem, it was shown in Cong and Ding [1994] that the optimal mapping depth is monotone.

3.3 Depth-Optimal Mapping Algorithm for *k/m*-macrocell

Nevertheless, we can have a depth-optimal mapping algorithm for k/m-macrocell. Given a cut (X, X') in N_v, the height of the cut, denoted as $h(X, X')$, is the maximum label in $input(X')$, that is,

$$h(X, X') = \max\{label(v) | v \in input(X')\}$$

It is assumed that every node in $input(X')$ already has a label. A min-height k/m-feasible cut (X, X') in a network is a k/m-feasible cut such that $h(X, X') \leq h(Y, Y')$, where (Y, Y') is any other k/m-feasible cut. Based on the definition of optimal mapping depth of v $label^*(v)$, we have $label^*(v) = h(X, X') + 1$, if (X, X')

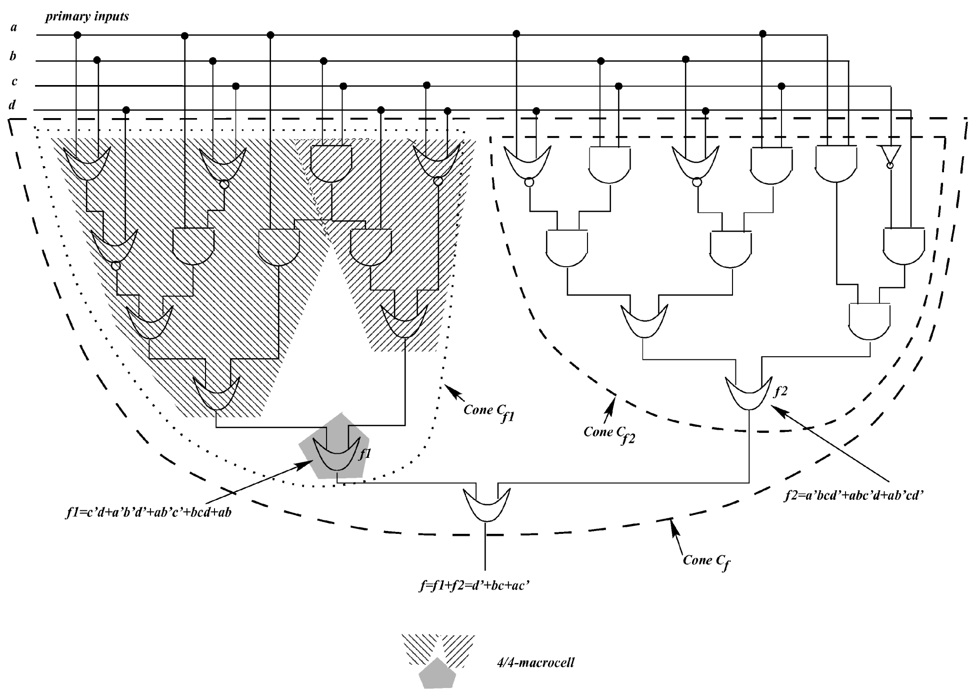


Fig. 1. An example showing the nonmonotone properties of the k/m-macrocell mapping problem.

is the min-height k/m-feasible cut in C_v and $label(u) = label^*(u)$ for any node u other than v in cone C_v. Therefore, following a similar argument as in Cong and Ding [1994], we can conclude that (i) a mapping algorithm can label every node v such that $label(v) = label^*(v)$ if it can find the min-height k/m-feasible cut for every node. And (ii) a mapping algorithm can find the depth optimal mapping solution for k/m-macrocell-based FPGAs if it can find the min-height k/m-feasible cut for every node.

Based on the above observation, a depth optimal mapping algorithm works as follows. It finds the min-height k/m-feasible cut for each node in the topological order from PIs to POs. It then can label each node v such that $label(v) = h(X, X') + 1 = label^*(v)$, where (X, X') is the min-height k/m-feasible cut for v. After labeling the whole network, it can use the min-height k/m-feasible cuts to generate the k/m-macrocells in the mapping solution. The mapping result has the optimal depth.

In order to find the min-height k/m-feasible cut for every node, we can exhaustively enumerate all k-feasible cuts and test if they are m-packable. The enumeration algorithm can then pick the k/m-feasible cut with minimum height for every node. However, such an algorithm is impractical to use due to the high complexity of exhaustive cut enumeration for a large k. In theory, the number of k-cuts of a node in a cone with n nodes can be as large as $O(n^k)$. Since we are interested in large k with values $k = 6$ to 10, we move to develop a more efficient heuristic algorithm.

3.4 The k_m_flow Algorithm—a Heuristic Approach

3.4.1 *Labeling Phase.* The k_m_flow algorithm also computes a label for each node from PIs to POs in topological order. At the beginning, every PI node will receive a label of 0. Then for every node v, suppose *mlevel* is the largest label among v's fanins, it is assumed that $label(v) \geq mlevel$. In order to test if we can set $label(v)$ to *mlevel*, we collapse each node u in N_v with $label(u) = mlevel$ into node v to form an induced network N'_v and test if we can find a k/m-feasible cut in N'_v. Obviously we assume that node labels will increase monotonically.

Based on the max-flow-min-cut theorem, we can easily find two min-cuts in N'_v: the max-volume-min-cut (X, X'), which is a min-cut with the largest $|X'|$, and the min-volume-min-cut (Y, Y'), which is a min-cut with the smallest $|Y'|$. Please note both (X, X') and (Y, Y') are min-cuts, which implies that $|input(X')| = |input(Y')| \leq |input(Z')|$ where (Z, Z') is any other cut in N'_v. Also, note that max-volume-min-cut and min-volume-min-cut are unique and $Y' \subseteq X'$. The max-volume-min-cut and min-volume-min-cut can be found in $O(ke)$ time, where e is the number of edges and k is the value of the maximum flow. There are three cases on whether the max-volume and min-volume min-cuts are k/m-feasible. m-packable is checked after calling *Espresso* [Brayton et al. 1984] on the functions in X' and Y' collapsed at v.

Case 1. Neither max-volume nor min-volume min-cut is k-feasible.

Because any k/m-feasible cut must be k-feasible too, this condition implies that no k/m-feasible cut exists in N'_v. In this case, node v can be simply labeled as $mlevel + 1$.

Case 2. Either max-volume or min-volume min-cut is k/m-feasible.

Suppose (X, X') is the k/m-feasible min-cut (either the max-volume or min-volume one). We can create a k/m-macrocell for node v, denoted as $map\_node(v)$, to implement the function of X'. The depth of $map\_node(v)$ in the mapping solution cannot be larger than *mlevel*. Therefore, we assign $label(v) = mlevel$.

Case 3. Both max-volume and min-volume min-cut are k-feasible but not m-packable.

Note that this condition does not guarantee that there is no k/m-feasible cut existing in N'_v. Therefore, we try to do a local cut enumeration in hope of finding a k/m-feasible cut.

To search for a k/m-feasible cut, perhaps the most natural way is to do a local cut enumeration within the cone defined by the max-volume-k-feasible-cut. However, unlike a max-volume-min-cut, the max-volume-k-feasible-cut may not be unique. Moreover, there is no good algorithm to find the max-volume-k-feasible-cut. Furthermore, it is intuitive to think that if the max-volume-min-cut (X, X') is not m-packable, a cut outside or across it may not likely be k/m-feasible, since it will have more fanins, tending to require more product terms in its sum-of-product representation. Therefore, in order to search for a k/m-feasible cut under Case 3, we only enumerate cuts inside X' to see if they are k/m-feasible.

To do a local cut enumeration in a cone C_v, first we mark all inputs to C_v as *pseudo-PIs* and then go through all nodes inside C_v in topological order from *pseudo-PIs* and enumerate all k-feasible cuts for every node v by the following equation, where x and y are the fanins of v[2]:

$$Cut(v) = (\{(C_x - x, x)\} \cup Cut(x)) \bigotimes_k (\{(C_y - y, y)\} \cup Cut(y)) \text{ [Cong et al. 1999].}$$

$Cut(x)$ is the set of k-feasible cuts for node x. The notation $(C_x - x, x)$ refers to the cut that cuts off the single node x. $\bigotimes_k$ is a merging operator defined on two cut sets; $S_1 \bigotimes_k S_2$ means merge every cut cut_1 in S_1 with every cut cut_2 in S_2 and only keep the k-feasible cuts in the result.

After the enumeration process, we check $Cut(v)$ to see if there is an m-packable cut. If there exists a k/m-feasible cut, node v can be labeled as *mlevel*; otherwise, it will be labeled as $mlevel + 1$.

The pseudocode for the labeling phase is shown in Figure 2. For all the nodes, we record the k/m-feasible cuts which correspond to their labels.

3.4.2 *Mapping Phase.* The second phase of our algorithm is to generate the k/m-macrocells in the mapping solution. For every node v, basd on the k/m-feasible cut (X, X') found in labeling phase, we can create a k/m-macrocell $map\_node(v)$ for v to implement the function of X' and $input(map\_node(v)) = input(X')$. The mapping phase is very similar to the mapping phase in FlowMap [Cong and Ding 1994]. The detailed description is shown in Figure 2.

3.4.3 *Properties of the k_m_flow Algorithm.* We can prove the following properties for the algorithm discussed above:

(1) If a node v is labeled as $label(v)$, it can then be implemented with a depth no more than $label(v)$. That is, $label(v)$ is the upper bound estimation of the depth of v in the mapping solution.
(2) If Case 3 never happens when mapping a specific circuit, then the mapping solution is delay optimal. Indeed, it is just the same as k-LUT mapping.
(3) For any certain circuit, if the optimal depth for k-LUT-based mapping is d_1, the optimal depth for k/m-macrocell-based mapping is d_2 and the depth of k_m_flow mapping result is d_3, then $d_1 \le d_2 \le d_3$.

3.5 Area Enhancement

After obtaining a k/m-macrocell mapping solution, we want to further reduce the number of k/m-macrocells in the mapping solution without increasing its depth.

[2]Although in problem formulation the given network is k-bounded, our algorithm is implemented for a two-bounded given network as it is proved in Cong and Hwang [1996] that any network can be fully decomposed into a two-bounded network without deteriorating the mapping quality. Therefore the cut enumeration equation provided here is for a two-bounded network. Of course it can be generalized to a k-bounded network at the cost of higher computation complexity.

```
Algorithm k_m_flow
BEGIN-k_m_flow
/* phase 1: labeling network */
for each PI node v do
  $label(v)$ := 0;
for each node $v$
{
  $mlevel$ := max { $label(n)$ — $n$ is a fanin of $v$ };
  collapse all nodes with $label$ = $mlevel$ into $v$ in $N_v$;
  find the max-vol-min-cut $(X, X^{'})$ for $v$;
  if (no cut exists) /* $N_v$ contains a single node */
  or if $(X, X^{'})$ is not $k$-feasible
    $label(v)$ := $mlevel + 1$;
  else
  {
    if ($X^{'}$ is $m$-packable)
      $label(v)$ := $mlevel$;
    else/* check if min-vol-min-cut is m-packable */
    {
      find the min-vol-min-cut $(Y, Y^{'})$ for $v$;
      if ($Y^{'}$ is $m$-packable)
        $label(v)$ := $mlevel$;
      else
      {
        mark all the inputs to cone $X^{'}$ as "pseudo-PIs";
        do a local cut enumeration starting from "pseudo-PIs";
        if find a $k/m$-feasible cut $(Z, Z^{'})$ through enumeration
          $label(v)$ := $mlevel$;
        else $label(v)$ := $mlevel + 1$;
      }   }
  }
}
/* phase 2: generating $k/m$-macrocells */
$L$ := list of PO nodes;
while $L$ contains non-PI nodes do{
  remove a non-PI node $v$ from $L$;
  let $(X, X^{'})$ be the cut generated by the labeling phase for node $v$;
  if $X^{'}$ contains only one node $v$ {
    generate a $k/m$-macrocell $map\_node$ to implement the function of the single node $v$; }
  else{
    generate a $k/m$-macrocell $map\_node$ to implement the
    function of $X^{'}$ such that $input(map\_node) = input(X^{'})$; }
    $L$ := $(L - \{v\}) \cup input(X^{'})$; }
END-k_m_flow
```

Fig. 2. Pseudocode of k_m_flow algorithm.

For every k/m-macrocell v, we try to pack as many of its predecessors with it as possible into a single k/m-macrocell. Clearly we need to guarantee the condition that the new k/m-macrocell is still k/m-feasible. In order to do so, we try to combine v with any of its fanins and then check if they can be packed into one k/m-macrocell. After v and one of its fanins have been successfully packed together, a new node v' will be formed to replace v in the mapping solution. It could be that some of the fanins of v' ($input(v') = input(v) \cup input(fanin)$) can still be packed together with v'. Therefore, the above greedy packing process will be repeated until no more nodes can be packed. The detailed packing algorithm, k_m_pack, is shown in Figure 3. On average, the total number of macrocells in the mapping solution may be reduced by a factor of 6% after the above packing process.

```
Algorithm k_m_pack
BEGIN-k_m_pack
$modified$:=true;
while $modified$==true do {
  $modified$:=false;
  for each fanin of $v$ do{
    if $|input(fanin) \cup input(v)| \leq k$ then {
      $v'$:=collapse fanin into $v$;
      if $v'$ has less than m product terms
        then {
          replace $v$ with $v'$;
          $modified$:=true;
          break; } } } }
END-k_m_pack
```

Fig. 3. Pseudocode of k_m_pack algorithm.

Table I. Description of 16 Benchmark Circuits

Circuit	duke2	C499	frg2	rot	ex4p	apex6	x3	apex5
#node	382	398	695	696	699	714	768	827
#PI	22	41	143	135	128	135	135	117
#PO	29	32	139	107	28	99	99	88
level	10	19	12	22	11	16	12	11
Circuit	i8	C2670	s5378	mm30a	pair	C3540	alu4	des
#node	917	1299	1317	1500	1556	2097	2347	3026
#PI	133	233	196	124	173	50	14	256
#PO	81	64	210	121	137	22	8	245
level	12	26	24	108	18	42	14	15

3.6 Experiment Result

Our algorithm, k_m_flow, has been implemented in C language within the Berkeley SIS and UCLA RASP [Cong et al. 1996] framework. We chose a set of 16 MCNC benchmarks to test k_m_flow on a Sun Ultra II workstation with 512M memory. Table I shows the size of the 16 benchmark circuits before mapping (all are two-bounded networks).

In order to find out the optimal mapping depth for each benchmark and compare it with the k_m_flow mapping solution, we implemented an algorithm called *k_m_enumerate*. The k_m_enumerate algorithm can find the depth-optimal mapping solution by performing an exhaustive cut enumeration in the entire network, as proposed in Section 3.3. We would like to point out that k_m_enumerate is impractical to use for a large k. We use it only to collect data to analyze the optimality of k_m_flow algorithm.

In Table II, we list the mapping depth generated by k_m_flow and k_m_enumerate under a different k and m. The data is in the form of x/y, where x is the depth of mapping solution generated by k_m_flow and y is the optimal mapping depth obtained by k_m_enumerate under the specified k and m. A question mark ? means the optimal depth is still unknown because of the extremely long runtime and large memory requirement of k_m_enumerate for a large k. From Table II, we can see that although k_m_flow cannot guarantee delay optimality in theory, in practice it is almost always able to find out the depth-optimal mapping solution.

Table II. Experimental Results Show that the k_m_flow Algorithm Can Achieve Depth-Optimal Mapping in Practice

	$k = 6$				$k = 8$				$k = 10$			
Circuit	$m = 6$	$m = 7$	$m = 8$	$m = 9$	$m = 8$	$m = 9$	$m = 10$	$m = 11$	$m = 10$	$m = 11$	$m = 12$	$m = 13$
duke2	4/4	4/4	4/4	4/4	3/3	3/3	3/3	3/3	3/3	3/3	3/3	3/3
C499	5/5	5/5	4/4	4/4	4/?	4/?	4/?	4/?	3/3	3/3	3/3	3/3
frg2	4/4	4/4	4/4	4/4	3/3	3/3	3/3	3/3	3/3	3/3	3/3	3/3
rot	6/6	6/6	6/6	6/6	5/5	5/5	5/5	5/5	4/4	4/4	4/4	4/4
ex4p	4/4	4/4	4/4	4/4	3/3	3/3	3/3	3/3	3/3	3/3	3/3	3/3
apex6	4/4	4/4	4/4	4/4	4/4	4/4	3/3	3/3	3/3	3/3	3/3	3/3
x3	3/3	3/3	3/3	3/3	3/3	3/3	3/3	3/3	2/2	2/2	2/2	2/2
apex5	4/4	4/4	4/4	4/4	3/3	3/3	3/3	3/3	3/3	3/3	3/3	3/3
i8	4/4	4/4	4/4	4/4	3/3	3/3	3/3	3/3	3/3	3/3	3/3	3/3
C2670	7/7	7/7	6/6	6/6	6/6	6/6	6/6	6/6	4/4	4/4	4/4	4/4
s5378	7/7	7/7	7/7	7/7	6/6	5/5	5/5	5/5	5/?	5/?	4/4	4/4
mm30a	21/21	21/21	21/21	21/21	20/20	20/20	20/20	17/?	19/?	15/?	15/?	15/?
pair	5/5	5/5	5/5	5/5	4/4	4/4	4/4	4/4	3/3	3/3	3/3	3/3
C3540	11/11	11/11	10/10	10/10	9/?	9/?	8/8	8/8	8/?	8/?	7/?	7/?
alu4	6/6	6/6	6/6	6/6	5/5	5/5	5/5	5/5	4/4	4/4	4/4	4/4
des	4/4	4/4	4/4	4/4	3/3	3/3	3/3	3/3	3/?	3/?	3/?	3/?

Table III. Total Mapping Depth of k/m-Macrocell Versus k-LUT on 16 MCNC Benchmarks

	Total depth for k/m-macrocell by k_m_flow				Total depth for k-LUT by FlowMap
	$m = k$	$m = k + 1$	$m = k + 2$	$m = k + 3$	
$k = 6$	99	99	96	96	95
$k = 7$	90	88	87	87	81
$k = 8$	84	83	82	78	75
$k = 9$	74	73	71	68	65
$k = 10$	73	70	67	67	62

Table IV. Total Number of k/m-Macrocells Versus Total Number of k-LUTs on 16 MCNC Benchmarks

	Total # of k/m-macrocells by k_m_flow				Total # of k-LUTs by FlowMap
	$m = k$	$m = k + 1$	$m = k + 2$	$m = k + 3$	
$k = 6$	7872	7728	7607	7554	7419
$k = 7$	6742	6574	6528	6496	6349
$k = 8$	6045	5971	5908	5909	5646
$k = 9$	5628	5526	5535	5513	5275
$k = 10$	5148	5128	5105	5075	4789

We also compare the k/m-macrocell mapping solution generated by k_m_flow with k-LUT mapping solution generated by FlowMap. Table III shows the total mapping depth of k/m-macrocell vs. k-LUT on the 16 benchmarks. Table IV shows the total number of macrocells vs. the total number of k-LUTs on 16 benchmarks. FlowMap is a depth-optimal k-LUT mapping algorithm based on flow computation. Since k/m-macrocells can be regarded as k-LUTs with additional m-product-term constraints, the optimal depth of a k-LUT mapping solution is the lower bound of the optimal depth of a k/m-macrocell mapping solution.

Table V. Quick Success Rate of Flow Computation in k_m_flow on 16 Benchmarks

	$m = k$	$m = k + 1$	$m = k + 2$	$m = k + 3$
$k = 6$	99%	99%	99.6%	99.8%
$k = 7$	97%	98%	99%	99%
$k = 8$	97%	98%	98%	99%
$k = 9$	96%	96%	97%	97%
$k = 10$	96%	96%	97%	97%

We can see that the mapping results of k_m_flow (both depth and number of macrocells) are close to the k-LUT mapping results when $m = k + 3$. It implies that the k/m-macrocell is almost equivalent to a k-LUT if m is slightly larger than k. This observation is consistent with the results reported by Kouloheris [1993 (p. 75)] where the author claimed the number of product terms needed to implement the function of a k-LUT grows almost linearly with k. Increasing the flexibility of the macrocell by allowing more product terms to be implemented will not significantly improve the performance.

The k_m_flow algorithm is a hybrid of flow computation and cut enumeration. The complexity of flow computation is $O(n^2)$ [Cong and Ding 1994]. The complexity of cut enumeration can be estimated as $O(npq^2)$,[3] where n is the number of nodes in the network, and p is *max{size of max-volume-k-feasible cone}*, and q is the max number of k-feasible cuts inside any cone. The equation assumes we will do a cut enumeration for every node in a cone of size p. The conservative estimation on the complexity to enumerate all k-feasible cuts for a single node is $O(q^2)$ because the input network is two-bounded. In order to get a polynomial complexity, we can bound the number of cuts enumerated by a constant number. Therefore, the complexity of k_m_flow is bounded by the complexity of flow computation $O(n^2)$.

The percentage of nodes where the max-volume or min-volume k-feasible cut returned by flow computation is m-packable is called *quick success rate*. *Quick success rate* is a characteristic of individual network and may differ from network to network. Fortunately, the quick success rate is on average 98% for $k = 6, 7, \ldots, 10$ and $m = k, k+1, \ldots, k+3$ of the 16 benchmarks. Detailed data is shown in Table V. It implies that k_m_flow has a complexity close to $O(n^2)$ in practice. It is understandable that due to this high success rate, k_m_flow will use almost the same cuts as FlowMap finds to create macrocells, resulting in the similar mapping depth.

From our observations on the range $k = 6$ to 10, the cones in which cut enumeration is performed are usually small, with less than 50 nodes inside. An exhaustive cut enumeration on a small network with no more than 50 nodes usually runs very fast. Therefore, the k_m_flow algorithm shall be an efficient algorithm to generate the k/m-macrocell mapping solution for medium k. For a large k, the cone may be large and even the local cut enumeration may take a long time to finish. Table VI shows the total CPU time (in seconds) needed to generate all the mapping solutions for 16 benchmarks. Since the *quick success*

[3]Assume cut enumertation is done on a two-bounded network.

Table VI. Total CPU Runtime of k_m_flow Algorithm on 16 Benchmarks

	$m = k$	$m = k + 1$	$m = k + 2$	$m = k + 3$
$k = 6$	105.9	101.7	96.8	96.1
$k = 7$	149.1	106.0	106.4	105.3
$k = 8$	119.0	119.0	118.7	119.3
$k = 9$	140.0	137.8	138.4	138.7
$k = 10$	210.5	207.8	206.3	207.6

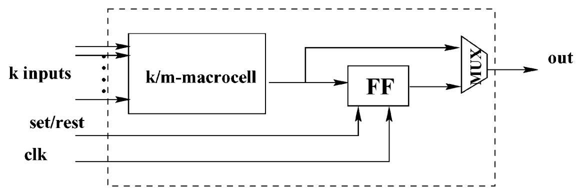


Fig. 4. k/m logic block.

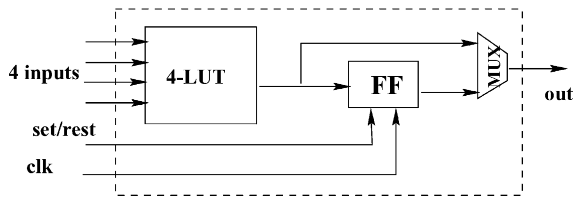


Fig. 5. 4-LUT logic block.

rate is usually very high, in practice, skipping local enumeration may cause little impact on the mapping quality and may save the runtime.

4. INVESTIGATION OF *k/m*-MACROCELL-BASED ARCHITECTURES

In Section 3 we used the unit area and the unit delay model to evaluate the quality of our k/m-macrocell mapping algorithm. In order to collect more accurate delay and area information to draw an architectural study conclusion, we use VPR [Betz et al. 1999], an FPGA placement and routing tool developed at the University of Toronto, to do placement and routing for our k/m-macrocell-based architecture and compare this architecture with the traditional 4-LUT-based architecture in terms of total area and critical path delay.

4.1 Area and Delay Models

Figure 4 shows the schematic diagram of the logic block used in our k/m-macrocell-based architecture (we call it the k/m the *logic block*). Figure 5 shows the logic block used in 4-LUT-based architecture [Betz et al. 1999] (we call it the 4-LUT *logic block*). Since the area of a logic block is greatly affected by the number of transistors in the basic cell, we use the ratio between the number of transistors in k/m-macrocell and that in 4-LUT to estimate the ratio of the area of the two logic blocks (i.e., k/m logic block vs. 4-LUT logic block). For the

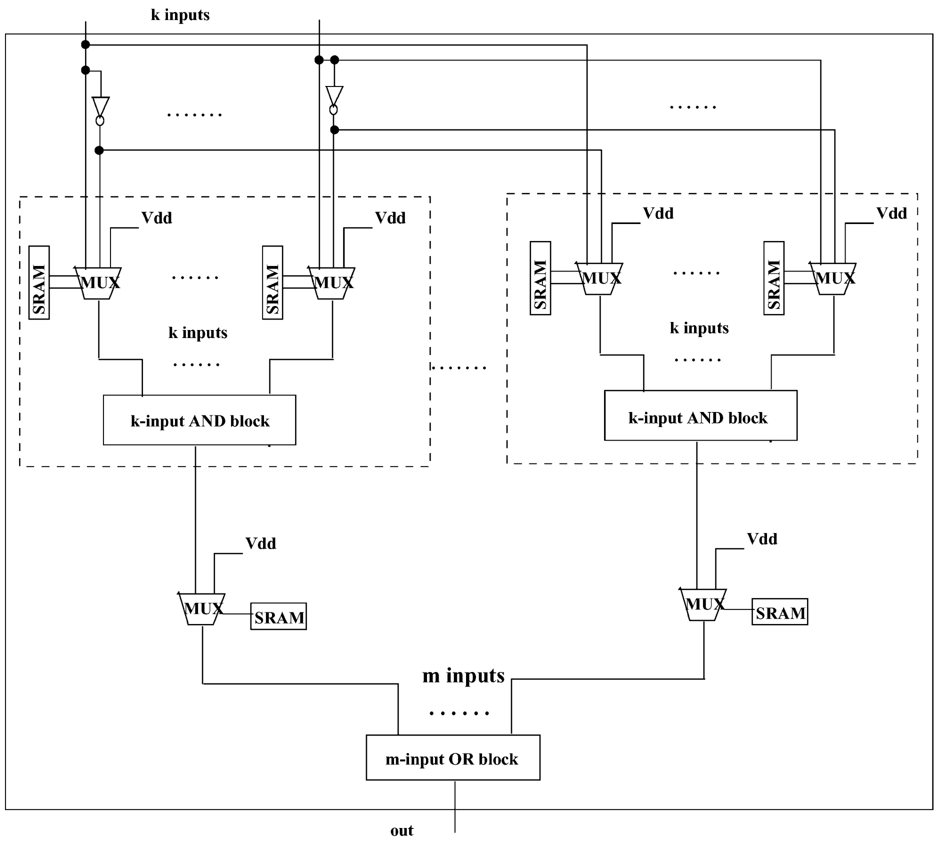
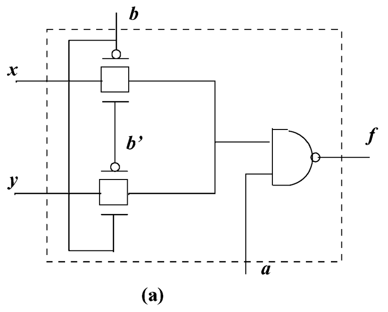


Fig. 6. Schematic diagram of k/m-macrocell.

logic block delay, we assume that it is proportional to the number of transistors on the longest path of the basic cell.

4.1.1 *Detailed Description of k/m-Macrocell Implementation.* Our k/m-macrocell consists of k inverters, km 3:1 MUXs, km 2-bit-SRAMs, m 1-bit-SRAM, km 2:1 MUXs, m k-input AND blocks, and one m-input OR block shown in Figure 6. The 3:1 MUX in the product-term block is used to choose x, x', and 1 for each input, so its logic function is $f = xab + yab' + a' = yb' + xb + a' = p' + a' = (pa)'$, $p = yb + xb'$. Therefore it can be implemented by the circuit shown in Figure 7(a), costing eight transistors. (We assume that SRAM can provide both data and its complementary, i.e., b' is provided.) There are three transistors on the longest path a signal would pass (one pass transistor and two n-transistors). The 2:1 MUX can be implemented by the circuit shown in Figure 7(b). It costs four transistors and there is one pass transistor in the longest path. The 1-bit-SRAM needs six transistors (see Betz et al. [1999], pp. 208, for the schematic diagram) and 2-bit-SRAM needs 12 transistors. The x-input AND block and y-input OR block can be implemented by multiple-level NAND gates and NOR gates. Here we adopt two-level gates to implement them



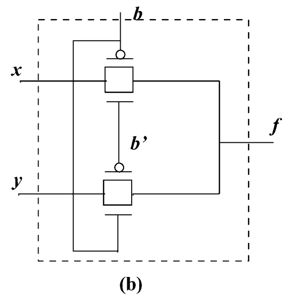


Fig. 7. 3:1 MUX and 2:1 MUX.

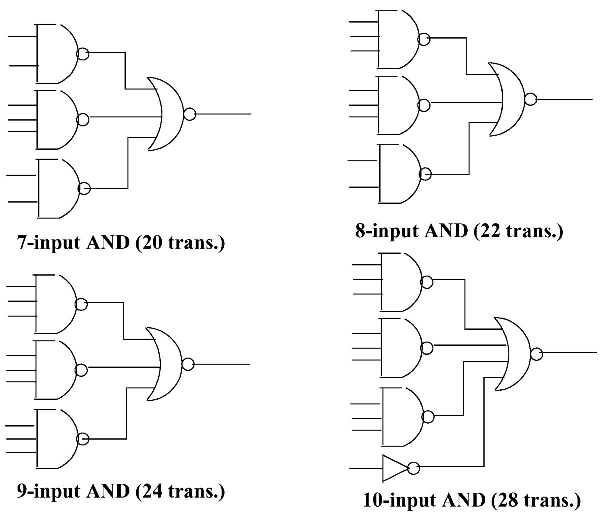


Fig. 8. x-input AND blocks.

as shown in Figures 8 and 9. The number of transistors used in the block is listed in the figures. For a z-input NAND (NOR) gate, there are z n-transistors (p-transistors) on the longest path; thus the number of transistors on the longest path of each x-input AND block and y-input OR block are listed as follows:

7-input AND block: $3 + 3 = 6$;	10-input OR block: $3 + 3 = 6$;
8-input AND block: $3 + 3 = 6$;	11-input OR block: $4 + 3 = 7$;
9-input AND block: $3 + 3 = 6$;	12-input OR block: $4 + 3 = 7$;
10-input AND block: $3 + 4 = 7$;	13-input OR block: $4 + 4 = 8$.

4.1.2 *Transistor Count for k/m-Macrocell Area Estimation.* The total transistor number of a k/m-macrocell is the sum of following items: k inverters ($2k$ transistors), km 2-bit-SRAM ($12km$ transistors), km 3:1 MUX ($8km$ transistors), m 2:1 MUX ($4m$ transistors), m 1-bit-SRAM ($6m$ transistors), m k-input AND block ($m\times$ (#transistors_per_k-input AND)), and 1 m-input OR block (#transistors_per_m-input OR), that is, $2k + 20km + 10m + m\times$ (#transistors_per_k-input AND)+#transistors_per_m-input OR.

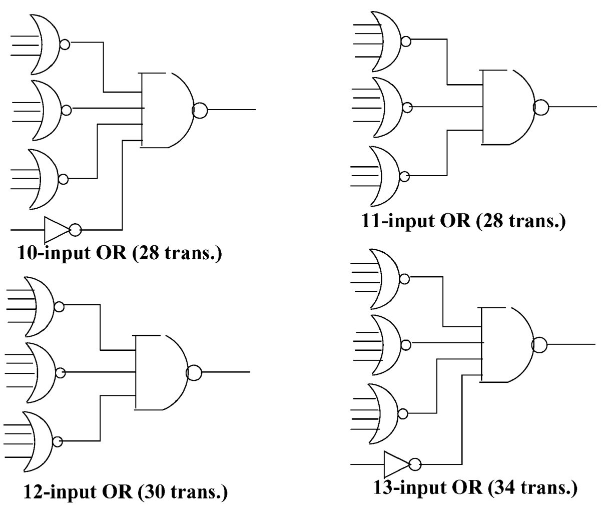


Fig. 9. x-input OR blocks.

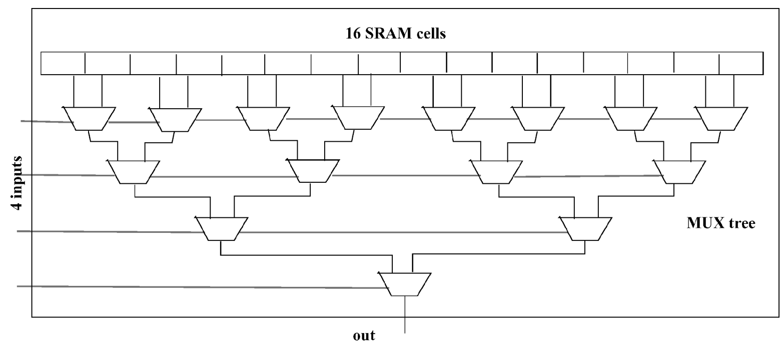


Fig. 10. Schematic diagram of 4-LUT-cell.

4.1.3 *Transistor Count for k/m-Macrocell Delay Estimation.* The longest path a signal would pass in the k/m-macrocell consists of an inverter, a 3:1 MUX, a k-input NAND block, a 2:1 MUX, and an m-input NAND block. Therefore the number of transistors in the longest path of k/m-macrocell is

$$\begin{array}{lll} k = 7, & m = 10: & 1+3+6+1+6 = 17; \\ k = 8, & m = 11: & 1+3+6+1+7 = 18; \\ k = 9, & m = 12: & 1+3+6+1+7 = 18; \\ k = 10, & m = 13: & 1+3+7+1+8 = 20. \end{array}$$

4.1.4 *Transistor Count for 4-LUT Area and Delay Estimation.* The 4-LUT cell we compare to is shown in Figure 10. Because we use four transistors to implement the 2:1 MUX, there should be extra four inverters for four inputs to provide their complementaries which are not drawn in this figure. This 4-LUT actually consists of 16 1-bit SRAMs, 4 inverters, and 15 2:1 MUXs. Therefore the total number of transistors used in this 4-LUT cell is $16\times6+15\times4+4\times2 = 164$.

Table VII. k/m Logic Block Area Estimation

	$k=7, m=10$	$k=8, m=11$	$k=9, m=12$	$k=10, m=13$
#Trans. of k/m-macrocell	1742	2156	2616	3148
#Trans. of k/m-macrocell vs. #Trans. of 4-LUT cell	10.6	13.2	16.0	19.2
Assumed delay ratio of k/m-macrocell vs. 4-LUT cell	3.4	3.6	3.6	4.0

The longest path a signal would pass is from an inverter to each level of 2:1 MUX; thus there are five transistors in the longest path.

Therefore, based on the above transistor counting and the estimation model for the logic block, we have the area and delay ratio of k/m-macrocell versus 4-LUT cell listed in the Table VII.

4.2 Experimental Setting of VPR

We used VPR version 4.22 to do placement and routing for the mapping solution of the k/m-macrocell-based architecture and that of the 4-LUT-based architecture. The authors of VPR extensively studied area/delay tradeoff for 4-LUT-based architecture. They proposed a detailed 4-LUT-based FPGA architecture under TSMC's 0.35-μm, 3.3-V process [Betz et al. 1999]. The 4-LUT logic block they proposed is identical to what Figure 5 shows. Two corresponding routing architecture files are attached with VPR package. In terms of interconnect configuration, one is simple and thus offers less area but a larger delay (we call it *RA1*), and the other is more complicated and offers a smaller delay but at a cost of larger area (we call it *RA2*). RA1 uses only buffer switched wires of length 1, while RA2 uses 50% buffer switched wires of length 4 and 50% pass transistor switched wires of length 4. As we only wanted to compare two architectures rather than getting an absolute result, it was reasonable to scale the area and delay parameters in the architecture file of 4-LUT-based architecture according to the area and delay estimation ratio between k/m logic block and 4-LUT logic block as discussed in Section 4.1 in order to derive the routing architecture file for k/m-macrocell-based architecture, that is, we compared our k/m-macrocell-based architecture with their 4-LUT-based architecture by only changing the parameters related to the area and delay of the logic block in the routing architecture files.

VPR reports routing area in the number of min-width transistors and the delay of critical path in seconds. We added up the logic block area to the routing area and got the total area of each mapping solution.

4.3 Experimental Results

We compared k/m-macrocell-based architecture with 4-LUT-based architecture by running VPR on their mapping solutions for the 16 MCNC benchmarks under the experimental settings mentioned above. The k/m-macrocell mapping solutions were obtained by running k_m_flow algorithm and then performing k_m_pack to further reduce the number of macrocells. The 4-LUT mapping solutions were obtained by running FlowMap followed by greedy-pack.

Table VIII. Normalized Area and Delay of k/m-Macrocell-Based Architecture on the 16 Benchmarks

	Results under routing architecture RA1			
	$k=7, m=10$	$k=8, m=11$	$k=9, m=12$	$k=10, m=13$
$A_{k/m\_tile}/A_{4-LUT\_tile}$	1.56	1.70	2.16	2.15
Total area	1.15	1.20	1.16	1.27
Crit. path delay	0.79	0.75	0.68	0.67
Routing area percentage	59%	56%	52%	48%
	Results under routing architecture RA2			
	$k=7, m=10$	$k=8, m=11$	$k=9, m=12$	$k=10, m=13$
$A_{k/m\_tile}/A_{4-LUT\_tile}$	1.46	1.59	1.47	2.03
Total area	0.85	0.86	0.81	0.83
Crit. path delay	0.86	0.81	0.74	0.73
Routing area percentage	71%	67%	63%	58%

Based on the estimation ratio discussed in Section 4.1, the average total area and critical path delay comparison between these two architectures are shown in Table VIII. A_{k/m_tile} refers to the area of the max-basic tile in k/m-macrocell-based architecture, which is the sum of the area of k/m logic block and the max-routing area per tile among the 16 benchmarks. A_{4-LUT_tile} refers to the area of the max-basic tile in 4-LUT-based architecture, which is the sum of the area of 4-LUT logic block and the max-routing area per tile among the 16 benchmarks. The area and delay of k/m-macrocell-based architecture were normalized with 4-LUT's =1. We compared the total area and critical path delay of the two architectures under the two routing architectures RA1 and RA2 mentioned in Section 4.2. We also reported the routing area percentage in k/m-macrocell-based architecture on the 16 benchmarks after routing.

For LUT-based FPGA, when k is small, most of the area is devoted to routing. With the increase of k, routing area decreases, but the area increase of logic blocks could be more than the decrease of routing area. Because the area of k/m-macrocell blocks does not grow exponentially as k-LUT does, the logic block area is much less than k-LUT-based architecture's; thus the total area decreases. Since the logic depth and routing area decrease, the total critical path delay decreases.

From the results, we can see that k/m-macrocell-based architecture can have 14–31% delay reduction with more area in RA1 and less area in RA2 when compared to 4–LUT-based architecture.

5. CONCLUSIONS AND FUTURE WORKS

We have studied a novel FPGA architecture based on k/m-macrocells through this article and proposed a k/m-macrocell technology mapping algorithm, named *k_m_flow*. In a set of 16 MCNC benchmarks, it produces optimal mapping depths in most test cases. Using this algorithm and this set of benchmarks, we have shown that k/m-macrocell-based FPGAs are similar to k-LUT-based FPGAs in terms of the mapping depths and the number of macrocells being used. The high quick success rate (Table V) suggests that k/m-macrocell can provide similar flexibility to a lookup table while each k/m-macrocell is much smaller than a k-LUT. We have analyzed the delays and areas of k/m-macrocell-based FPGAs using VPR on this set of benchmarks. We compared the results with those of traditional 4-LUT-based FPGAs. Our comparison has shown that k/m-macrocell-based FPGAs can significantly outperform 4-LUT-based FPGAs in terms of delay.

Future works includes two directions. (i) We can use a packing process to pack mapped k/m-macrocells into multiple-output macrocells in order to support CPLDs mapping. (ii) As the area and delay models we used for k/m-macrocells in Section 4 are primitive, a more accurate and detailed model needs to be studied.

ACKNOWLEDGMENTS

The authors would like to thank Professor David Lewis and Dr. Vaughn Betz for their helpful comments and discussion.

REFERENCES

ALTERA CORP. 2000. *MAX 700B, Programmable Logic Device Family*. San Jose, CA. Web site: www.altera.com.

ALTERA CORP. 2001. *APEX II, Programmable Logic Device Family*. San Jose, CA. Web site: www.altera.com.

ANDERSON, J. H. AND BROWN, S. D. 1998. Technology mapping for large complex PLDs. In *Proceedings of the Design Automation Conference*. 698–703.

BETZ, V., ROSE, J., AND MARQUARDT, A. 1999. *Architecture and CAD for Deep-Submicron FPGAs*. Kluwer Academic, Norwell, MA.

BRAYTON, R., SANGIOVANNI-VINCENTELLI, A., HACHTEL, G., AND MCMULLIN, C. 1984. *Logic Minimization Algorithms for Digital Circuits*. Kluwer, Norwell, MA.

CONG, J., CHEN, D., ERCEGOVAC, M., AND HUANG, Z. 2001. Performance-driven mapping for CPLA architecture. In *Proceedings of the ACM International Symposium on FPGA*. 39–47.

CONG, J. AND DING, Y. 1994. An optimal technology mapping algorithm for delay optimization in lookup-table based FPGA designs. *IEEE Trans. Comput.-Aid. Des. Integr. Circ. Syst. 13*, 1, 1–12.

CONG, J. AND DING, Y. 1996. Combinational logic synthesis for LUT based field programmable gate arrays. *ACM Trans. Des. Automat. Electron. Syst. 1*, 2, 145–204.

CONG, J., HUANG, H., AND YUAN, X. 2000. Technology mapping for k/m-macrocell based FPGAs. In *Proceedings of the ACM International Symposium on FPGA*. 51–59.

CONG, J. AND HWANG, Y. 1996. Structural gate decomposition for depth-optimal technology mapping in LUT-based FPGA. In *Proceedings of the Design Automation Conf.* 726–729.

CONG, J., PECK, J., AND DING, Y. 1996. RASP: A general logic synthesis system for SRAM-based FPGAs. In *Proceedings of the ACM International Symposium on FPGA*. 137–143.

CONG, J., WU, C., AND DING, Y. 1999. Cut ranking and pruning: Enabling a general and efficient FPGA mapping solution. In *Proceedings of the ACM International Symposium on FPGA*. 29–35.

CYPRESS SEMICONDUCTOR CORP. 2000. *The Cypress Data Book*. San Jose, CA. Web site: www.cypress.com.

FRANCIS, R. J., ROSE, J., AND VRANESIC, Z. G. 1991a. Chortle-CRF: Fast technology mapping for lookup table-based FPGAs. In *Proceedings of the Design Automation Conference*. 227–233.

FRANCIS, R. J., ROSE, J., AND VRANESIC, Z. G. 1991b. Technology mapping of lookup table-based FPGAs for performance. In *Proceedings of the International Conference on Computer Aided Design*. 568–571.

HASAN, Z., HARRISON, D., AND CIESIELSKI, M. 1992. A fast partition method for PLA-based FPGAs. *IEEE Des. Test Comput. 9*, 4, 34–39.

KAVIANI, A. AND BROWN, S. 1999. The hybrid field-programmable architecture. *IEEE Des. Test Comput. 16*, 74–83.

KAVIANI, A. S. 1999. Novel architecture and synthesis methods for high capacity field programmable devices. Ph.D. dissertation. University of Toronto, Toronto, Ont., Canada.

KOULOHERIS, J. L. 1993. Empirical study of the effect of cell granularity on FPGA density and performance. Ph.D. dissertation. Stanford University, Stanford, CA.

KOULOHERIS, J. L. AND GAMAL, A. E. 1992. PLA-based FPGA area versus cell granularity. In *Proceedings of the IEEE Custom Integrated Circuits Conference*. 4.3/1–4.

KRISHNAMOORTHY, S. AND TESSIER, R. May 2003. Technology mapping algorithms for hybrid FPGAs containing lookup tables and PLAs. *IEEE Trans. Comput.-Aid. Des. Integr. Circ. Syst. 22*, 5, 545–559.

LATTICE SEMICONDUCTOR CORP. 2000. *The Lattice Data Book*. Hillsboro, OR. Web site: www.latticesemi.com.

ROSE, J., FRANCIS, R. J., LEWIS, D., AND CHOW, P. Ct. 1990. Architecture of field programmable gate arrays: The effect of logic block functionality on area efficiency. *IEEE J. Solid State Circ. 25*, 5, 1217–1225.

XILINX INC. 2001. *Virtex-II Field-Programmable Gate Arrays*. San Jose, CA. Web site: www.xilinx.com.

Received April 2003; revised August 2003; accepted January 2004

Editorial

Welcome to a new issue of ACM TODAES, and a very Happy New Year to all readers!

I started as the new Editor-In-Chief (EIC) of TODAES in August 2004, having taken over from the previous EIC, Prof. Mary Jane Irwin. Prof. Irwin provided outstanding leadership and worked tirelessly for the journal during her tenure: she handled the difficult (and thankless!) task of transitioning from the manual submission process to the current electronic scheme (ACM Manuscript Central) in January 2003; she diligently monitored the growing backlog of accepted articles; she sought additional funds from SIGDA to publish oversized "phonebook" issues to accommodate and eliminate the backlog of additional accepted articles when we were constrained by ACM's annual page limit budget; she solicited tutorial articles from top researchers to cover state-of-the-art in new CAD topics.... I could go on and fill this whole page, but it suffices to say that I am stepping into some pretty big shoes! We owe Prof. Irwin a big thank you for her dedicated service as the EIC of TODAES from 1998 through 2004. We also thank Susan Benner for her help as the Editorial Assistant under EIC Irwin. Melanie Sanders has taken over as Editorial Assistant for ACM TODAES.

With this issue, our roster of Associate Editors (AEs) has also undergone some changes. AEs Steve Levitan and Irith Pomeranz have retired and we thank them for their efforts in handling articles related to system design and testing/reliability. I am pleased to announce the addition of four new AEs: Prof. Tim Cheng (University of California, Santa Barbara), Prof. Sharon Hu (University of Notre Dame), Prof. Sharad Malik (Princeton University), and Professor Hans-Joachim Wunderlich (University of Stuttgart). Together, they both broaden and strengthen the technical reach of ACM TODAES in the areas of testing, real-time CAD, platform-based design, and formal methods. Please join me in welcoming them aboard ACM TODAES!

I have also appointed Prof. Prabhat Mishra (University of Florida) to serve as Information Director for ACM TODAES. He has updated the TODAES Web page and you can see the results at http://www.acm.org/todaes/.

Several changes and clarifications in the policies for ACM TODAES are being developed, including policies for submission of manuscripts that are extended versions of archival conference/workshop articles, recommended lengths for manuscripts, goals for improving turnaround time, invitation of tutorial/survey articles, special issues of the journal, etc. These changes and updates will be reflected on the TODAES Web site, so please visit the Web site and familiarize yourself with the updated policies before submitting a manuscript for review.

I welcome feedback on any issues you've encountered with ACM TODAES, and, of course, suggestions for how we might improve ACM TODAES. I look forward to starting my term as EIC of TODAES and solicit technical contributions that will expand the quality, reach, and visibility of ACM TODAES to make the journal a valuable archival resource for the entire electronic design automation community.

NIKIL DUTT
Editor-In-Chief

Bipartitioning and Encoding in Low-Power Pipelined Circuits

SHANQ-JANG RUAN
National Taiwan University of Science and Technology
KUN-LIN TSAI
National Taiwan University
EDWIN NAROSKA
University of Dortmund
and
FEIPEI LAI
National Taiwan Univerisity

In this article, we present a bipartition dual-encoding architecture for low-power pipelined circuits. We exploit the bipartition approach as well as encoding techniques to reduce power dissipation not only of combinational logic blocks but also of the pipeline registers. Based on Shannon expansion, we partition a given circuit into two subcircuits such that the number of different outputs of both subcircuits are reduced, and then encode the output of both subcircuits to minimize the Hamming distance for transitions with a high switching probability. We measure the benefits of four different combinational bipartitioning and encoding architectures for comparison. The transistor-level simulation results show that bipartition dual-encoding can effectively reduce power by 72.7% for the pipeline registers and 27.1% for the total power consumption on average. To the best of our knowledge, it is the first work that presents an in-depth study on bipartition and encoding techniques to optimize power for pipelined circuits.

Categories and Subject Descriptors: B.6.1 [**Logic Design**]: Design Styles—*Combinational logic*; B.6.3 [**Logic Design**]: Design Aids—*Automatic synthesis*; J.6 [**Computer-Aided Engineering**]: *Computer-aid design (CAD)*

General Terms: Algorithms, Design

Additional Key Words and Phrases: Low-power design

This work was partially supported by NSC92-2218-E-011-066.
Authors' addresses: S.-J. Ruan, Department of Electronic Engineering, National Taiwan University of Science and Technology, Taipei, Taiwan 106; email: sjruan@et.ntust.edu.tw; K.-L. Tsai, Department of Electrical Engineering, National Taiwan University, Taipei, Taiwan 106; email: kunlin@orchid.ee.ntu.edu.tw; E. Naroska, Computer Engineering Institute, University of Dortmund, 44221 Dortmund, Germany; email: edwin@ds.e-technik.uni-dortmund.de; F. Lai, Department of Computer Science and Information Engineering & Department of Electrical Engineering, National Taiwan University, Taipei, Taiwan 106; email: flai@cc.ee.ntu.edu.tw.

1. INTRODUCTION

In modern processor designs, pipelining is the most popular fashion to increase overall performance. Since Alidina et al. [1994] first applied precomputation on sequential logic to achieve low power, much work has been published on synthesis for low-power pipelined CMOS circuits. We categorize this previous work into three groups.

The first group covers approaches where pipeline registers are immovable. Techniques like precomputation [Alidina et al. 1994], gated pipeline registers [Ye and Irwin 1999; Kapadia et al. 1999], and guarded evaluation [Tiwari et al. 1998; Munch et al. 2000] belong to this category. Unfortunately, as for these approaches registers are immovable, further improvements are limited. Another limitation is that they require additional control logic.

In the second category, pipeline registers are movable. Approaches like retiming [Moterio et al. 1993] and repositioning of registers in datapaths [Schimpfle et al. 1997] belong to this category. However, no effort is made to modify the combinational logic blocks.

Approaches of the third category reduce the size of active registers and logic blocks using partition techniques [Choi and Hwang 1999]. Choi and Hwang [1999] partitioned the combinational logic block of a pipelined circuit into multiple subcircuits by recursively applying Shannon expansion with respect to the selected input variables. Furthermore, Ruan et al. [1999, 2001] showed that partitioning circuits into more than two sections does not always save power due to the overhead of duplicated input registers and output multiplexors.

Some preliminary work on combining techniques of bipartitioning and retiming has been done in Ruan et al. [1999] and Chen et al. [2001] , but the inability to extract the most active portion may make this approach inapplicable in the real world. In this article, we take advantage of bipartitioning and encoding techniques toward optimizing power consumption of pipelined circuits. As in Ruan et al. [1999] and Chen et al. [2001], we consider a pipeline architecture where combinational logic blocks are separated by edge-triggered registers that are driven by a single clock signal. We propose a bipartition dual-encoding architecture to decrease power consumption of pipelined CMOS designs. Our approach is based on the observation that the pipeline registers take a large fraction of total power dissipation for most of the circuits. Table I shows that in our experiments pipeline register account for 64.6% of the total power budget on average. In order to address this issue, we first bipartition a given circuit by using Shannon expansion to minimize the number of different outputs of both subcircuits [Micheli 1994]. Second, we encode both partitions to reduce the switching activities of the pipeline registers and logic blocks. To validate the results, we employ the accurate transistor-level power estimator EPIC PowerMill[1] to estimate power dissipation.

The rest of the article is organized as follows. In Section 2, we present bipartition and bipartition single-encoding architectures and discuss their characteristics. Further, our new bipartition dual-encoding architecture is

[1]EPIC PowerMill was developed by EPIC Design Technology, Inc.

Table I. Power Dissipation of Registers for Several MCNC Benchmarks

Circuits	sao2	9sym	con1	misex1	rd53	rd73	rd84	sqrt8	xor5	t481	Ave.
Reg.%	59.2	42.2	87.4	74.7	68.5	62.1	49.9	64.6	81.4	91.0	64.6

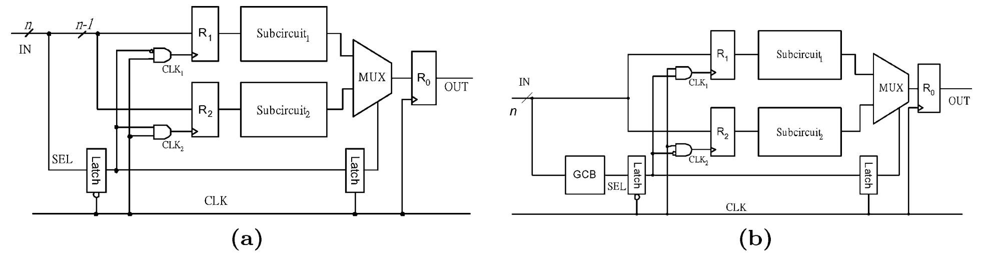


Fig. 1. First bipartition architecture (a) and bipartition architecture based on output extraction (b).

presented. The synthesis algorithms for the proposed architecture are presented in Section 3. The experimental results and conclusions are given in Sections 4 and 5.

2. BIPARTITIONING AND ENCODING ARCHITECTURES

In Figure 1(a), a bipartition architecture based on Shannon expansion [Choi and Hwang 1999] is shown. Depending on the value of *SEL*, only one of the subcircuits is active while the other is disabled. Power saving is achieved if each of the two subcircuits consumes less power than a direct implementation. The disadvantage of this architecture is that duplicated registers always increase the area overhead and limit power saving.

Figure 1(b) shows the second bipartition architecture. Compared to Figure 1(a), the *SEL* signal is generated by *GCB* (global control block). The basic concept of this approach is to assign a few but frequently occurring outputs to form a (small) module $Subcircuit_1$ while the remaining (less frequently occurring) outputs are moved to $Subcircuit_2$ (see Figure 1(b)). Depending on the activity of the outputs, the architecture can have significant power reduction, even though the duplicated registers and the selection logic (GCB) may incur area and power overheads.

Starting from Figure 1(b), we now replace the highly active $Subcircuit_1$ with an encoder-decoder (codec) architecture to reduce the internal switching activity of R_1 and logic block (Figure 2(a)). Notice that *Encoder* not only encodes the k frequently occurring output pattern with minimal Hamming distance but also generates the selection signal (*SEL*). The interested reader may refer to Ruan et al. [1999] for a further analysis on this topic.

Based on the previous architectures, we propose a new bipartition dual-encoding approach for lowpower pipelined circuits. The main idea of this approach is to partition circuits using Shannon expansion for simplifying selection logic. Then we encode both subcircuits to reduce size as well as switching activity of registers. In Figure 2(b), the bipartition dual-encoding architecture

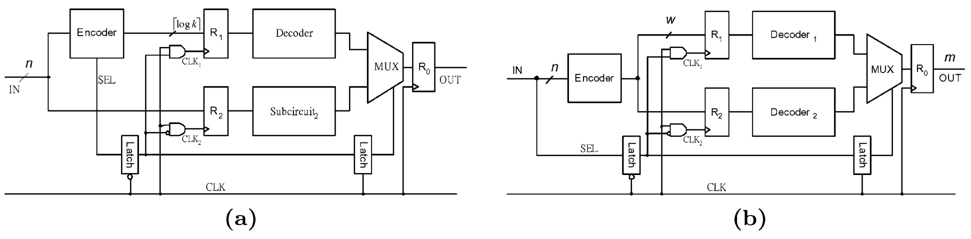


Fig. 2. A bipartition single-encoding (a) and dual-encoding (b) architecture.

is shown. Only one of the input signals is selected as partition variable *SEL* but all input signals feed into the *Encoder*, which encodes the output with minimal register size and Hamming distance. Depending on *SEL*, either R_1 and $Decoder_1$ or R_2 and $Decoder_2$ are activated.

3. SYNTHESIS OF THE BIPARTITION DUAL-ENCODING ARCHITECTURE

3.1 Bipartition Algorithm

The choice of partition variable *SEL* is critical in the bipartition dual-encoding architecture. To find a suitable solution, we use a brute force approach to test and rate all (i.e., n, where n is the number of input pins) possible configurations. This approach is acceptable as n is usually small (less than 100).

In detail, each variable is selected as partition variable and the PLA is partitioned accordingly. Based on the partitioned PLA table, the number of different output pattern are determined for $partition_1$ and $partition_2$. These numbers are denoted as OP_1 and OP_2, respectively, in the following. Next, the configuration is rated according to $w = \max(\lceil \log_2(OP_1) \rceil, \lceil \log_2(OP_2) \rceil)$. Finally, the configuration which gives a minimal rating w is selected as the final bipartitioning result.

3.2 Encoding Algorithm

As total power dissipation mainly depends on the switching activity of the pipeline registers, we try to encode the output pattern so that the hamming distance between pattern with a high transition probability is minimal. Usually this will also have a positive effect on the power consumed by the combinational logic blocks $Decoder_1$ and $Decoder_2$.

The encoding problem consists of choosing codes for the outputs of both sub-PLAs that minimize the switching probability. As described in Section 3.1, two sub-PLAs, PLA_1 and PLA_2, are obtained after bipartitioning. The number of output patterns of PLA_1 and PLA_2 are denoted as OP_1 and OP_2, respectively. Therefore, the output bit width of the encoder is $\max(\lceil \log_2 OP_1 \rceil, \lceil \log_2 OP_2 \rceil)$. The *PLA* with the maximum number of different outputs is denoted as PLA_x in the following. Next, we adopt the heuristic algorithm introduced in Benini and Micheli [1995] to encode PLA_x. The other *PLA* (denoted as PLA_y) is encoded using the output pattern of PLA_x as follows: the (not encoded) output patterns of both *PLA*s are sorted in decreasing order of their occurrence frequency.

The sorted patterns are denoted as *sort*(PLA_x) and *sort*(PLA_y). Further, the encoded values are also sorted according to *sort*(PLA_x). Finally, encoding of PLA_y is determined by assigning the first pattern of *sort*(PLA_y) to the first pattern from the sorted encoding list, and so on.

Example. Consider an original combinational block with the following truth table:

input $x_0x_1x_2$	000	001	010	011	100	101	110	111
output $y_0y_1y_2$	000	000	001	001	111	111	111	010

If we choose x_0 as partition variable SEL, PLA_1 and PLA_2 become:

PLA_1			PLA_2		
SEL ($= x_0$)	x_1x_2	$y_0y_1y_2$	SEL ($= x_0$)	x_1x_2	$y_0y_1y_2$
0	00	000	1	00	111
0	01	000	1	01	111
0	10	001	1	10	111
0	11	001	1	11	010

From the truth tables we determine the encoder output width to be 1 bit (each sub-PLA uses only two different output patterns). From PLA_1 and PLA_2 we determine the output frequency of each pattern:

PLA_1		PLA_2	
$y_0y_1y_2$	frequency	$y_0y_1y_2$	frequency
000	2	111	3
001	2	010	1

As both *PLA*s are having the same number of different outputs (2) we randomly choose PLA_1 to be encoded first. Assume that pattern output 000 is encoded as 0 and 001 is encoded as 1. As a result, we get the following assignments:

PLA_1			PLA_2		
$y_0y_1y_2$	frequency	encoding	$y_0y_1y_2$	frequency	encoding
000	2	0	111	3	0
001	2	1	010	1	1

Note that the encoding column for PLA_2 is obtained by simply copying the *encoding* column of PLA_1. Hence, the truth table for the encoder is

$x_0x_1x_2$	000	001	010	011	100	101	110	111
encoded outp.	0	0	1	1	0	0	0	1

As a result, the truth tables for $Decoder_1$ and $Decoder_2$ are

$Decoder_1$		$Decoder_2$	
encoded input	$y_0y_1y_2$	encoded input	$y_0y_1y_2$
0	000	0	111
1	001	1	010

Compared to an optimal output (state) assignment approach, the bipartitioning technique may deliver additional switching reduction. For example, assume that there are four outputs a, b, c, and d with probability $p_a = 0.7$, $p_b = p_c = p_d = 0.1$ (output transitions takes place randomly and independent from the current output value). Optimal assignment requires 2 output bits, for example, with assignments $a = 00$, $b = 01$, $c = 10$, and $d = 11$. As a result, on average 0.64 state bits will flip from clock cycle to clock cycle (note that a "transition" from d to d is also permitted).

Now assume that we bipartition the output space so that a and b are assigned to partition P_1 and c and d are assigned to partition P_2. P_1 as well as P_2 now can be encoded with 1 bit. We assume that a and c are assigned value 0, b and d are assigned 1. Note that we need a register bit to store which of the partitions is active. Hence, we actually have three register bits: one for P_1, one for P_2, and one to identify the active partition. As a result, the total number of bits flipping for all three registers will sum up to 0.595 on average. Compared to the previous result (0.64) the difference is due to the fact, that P_1 stores value a (probability 0.7) most of the time (i.e., the register stores 0). If there is a transition from state c or d (belongs to P_2) to a (belongs to P_1), then there will be a good chance that the register of P_1 already stores 0 ($= a$) and, hence, does not flip.

4. EXPERIMENTAL RESULTS

We implemented the algorithm and applied it to some MCNC benchmark circuits. We used the SIS[2] standard script *script.rugged* to obtain a multilevel implementation of *Encoder*, *Decoder*$_1$, and *Decoder*$_2$ for TSMC[3] 0.25-μm technology. The modules were then integrated by adding control elements latches, registers, AND gates, and multiplexers. The integrated implementations were simulated at transistor level (EPIC PowerMill) applying equally distributed random patterns to the inputs. Supply voltage and clock frequency were set to 2.5 V and 20 MHz. The area unit and power unit were μm^2 and μW, respectively, throughout the section. The power reduction rate and area increase rate were computed as $100(P_{orig} - P_{proposed})/P_{orig}$ and $100(A_{proposed} - A_{orig})/A_{orig}$, respectively, throughout the experiments.

Power dissipation of pipeline registers for original, bipartition architecture and bipartition dual-encoding architecture are named as Orig, Bipart, Bi_dual in Table II. The columns PF_B% and PF_Dual represent the power reduction of bipartition (based on Shannon expansion; Figure 1(a)) and bipartition dual-encoding architectures, respectively. Experiments showed that our bipartition algorithm dissipated 23.9% fewer power compared to the original architecture in pipeline registers. Further, we obtained a significant power reduction of 72.7% by using the bipartition dual-encoding architecture.

Table III presents the performance of our bipartition algorithm (Figure 1(a)). The "Original" columns show the power dissipation of combinational block "P_{block}" and total area "Area." In the "Bipartition architecture" columns, the power dissipation of modules *subcircuit*$_1$, *subcircuit*$_2$, and multiplexors (see Figure 1(a)) are denoted as "Subc$_1$," "Subc$_2$," and "Mux," respectively. The power dissipation of Clock covers both AND gates and latches. Finally, power improvement and area increase are labeled as "PR%" and "AI%," respectively. The

[2]SIS: A system for sequential circuit synthesis implemented by the Electronic Research Laboratory in the Department of EE and CS, University of California, Berkeley, on May 4, 1992.
[3]TSMC stands for Taiwan Semiconductor Manufacturing Company.

Table II. Register Power Dissipations of the Original Circuit and the Bipartition Dual-Encoding Architecture

Circuits	Orig	Bipart	Bi_dual	PF_B%	PF_Dual%
sao2	548.7	207.9	81.7	62.1	85.1
9sym	489.5	424.6	51.6	13.3	89.5
con1	348.8	276.0	115.4	20.9	66.9
misex1	407.3	254.4	152.2	37.6	62.6
rd53	263.9	200.5	134.5	24.0	49.0
rd73	368.0	247.4	152.4	32.8	58.6
rd84	408.6	350.1	171.4	14.3	58.0
sqrt8	407.9	355.1	182.9	12.9	55.1
xor5	248.5	192.7	64.2	22.5	74.2
t481	767.9	733.8	56.3	4.44	92.7
Average	452.9	324.2	116.3	23.9	72.7

Table III. Simulation Result of Original Circuit and Bipartition Architectures Based on Shannon Expansion

	Original		Bipartition Architecture							
Design	P_{block}	Area	$Subc_1$	$Subc_2$	Clock	Mux	Total	Area	PR%	AI%
sao2	406.0	2954.9	178.1	0.0	124.2	24.6	534.8	2995.2	44.0	1.4
9sym	694.3	3553.9	166.6	167.2	129.6	9.4	897.3	4389.1	24.2	23.5
con1	58.4	929.3	28.2	14.9	119.7	19.5	458.2	1799.0	−12.5	93.6
mixex1	153.7	1509.1	33.7	49.0	125.6	68.5	531.2	2580.5	5.3	71.0
rd53	135.9	956.2	43.4	56.0	114.8	34.0	448.7	1877.8	−12.2	96.4
rd73	246.9	1658.9	114.8	113.3	128.1	46.8	650.4	3162.2	−5.8	90.6
rd84	447.3	2494.1	242.4	122.4	131.8	56.5	903.2	4026.2	−5.5	61.4
sqrt8	254.7	1687.7	42.9	68.2	126.9	39.8	632.8	2603.5	4.9	54.3
xor5	70.6	691.2	27.7	14.7	109.6	16.2	360.9	1336.3	−13.1	93.3
t481	97.1	1791.4	36.2	58.1	166.0	11.0	1005.1	4039.7	−16.2	125.5
Average	256.8	1822.7	91.4	66.4	127.6	32.6	642.3	2881.0	5.9	58.1

results show that the bipartition architecture suffers from the power dissipated by Clock and Mux.

Table IV shows power and area numbers for the original circuits as well as for the bipartition dual-encoding architecture. The columns in this table have the same meaning as in Table III, except for the second column which shows total power dissipation of the original circuit. The columns "Enc," "Dec_1," and "Dec_2" stand for the power dissipated by the corresponding blocks Encoder, $Decoder_1$, and $Decoder_2$, respectively, from Figure 2(b). The results show that the power saving effects could be increased from 5.9% to 27.1% while the area overhead was reduced from 58.1% to 3.4% compared to the plain bipartition technique indicated in Table III.

Table V shows the average area increase and power reduction of bipartition based on Shannon expansion (Bipart-c [Ruan et al. 2001]), bipartition based on output clustering (Bipart-s), bipartition single-encoding (Bipart-single [Ruan et al. 2001]), and bipartition dual-encoding architecture (Bipart-dual) for comparison. The data of the first and third columns are cited from Ruan et al. [2001]. The columns "AI%," "PF%", and "PR%" represent the area increase,

Table IV. Simulation Result of Original Circuit and Bipartition Dual-Encoding Architectures

	Original		Bipartition dual-encoding architecture								
Design	Power	Area	Enc	Dec_1	Dec_2	Clock	Mux	Total Power	Area	PR%	AI%
sao2	954.7	2954.9	422.4	13.8	0.0	113.0	22.9	653.8	2849.3	31.5	−3.6
9sym	1183.8	3553.9	474.4	0.0	0.0	64.9	37.9	628.8	2327.0	46.9	−34.5
con1	407.2	929.3	105.0	5.3	5.6	108.3	23.3	362.9	1146.2	10.9	23.3
mixex1	561.0	1509.1	117.7	13.1	36.4	118.1	65.6	503.1	2016.0	10.3	33.6
rd53	399.8	956.2	72.7	15.8	9.8	106.5	29.6	369.0	1561.0	7.7	63.3
rd73	614.9	1659.0	230.9	17.5	13.7	111.3	34.1	559.9	2181.1	9.0	31.5
rd84	855.9	2494.1	400.7	8.4	14.2	114.6	36.0	745.3	2663.0	12.9	6.8
sqrt8	665.3	1687.7	253.7	19.7	40.8	77.6	79.3	654.2	2488.3	1.7	47.4
xor5	319.1	691.2	53.9	1.0	0.8	95.3	9.9	225.1	616.3	29.5	−10.8
t481	865.0	1791.4	113.2	0.0	0.0	98.3	10.1	278.0	996.4	67.9	−44.4
Average	682.7	1822.7	224.5	9.5	12.1	100.8	34.9	498.0	1884.5	27.1	3.4

Table V. Average Area and Power Comparison Between Single and Dual Encoding

	Bipart-c	Bipart-s	Bipart-single	Bipart-dual
AI%	44.4	58.1	29.6	3.4
PF%	26.0	23.9	63.0	72.7
PR%	9.7	5.9	31.6	27.1

power reduction of pipeline registers, and total power reduction, respectively. As shown in this table, bipartition dual-encoding architecture obtains the significant power saving in pipeline registers (PF%); however, the overall power saving is a little less than that of the single-encoding architecture. This is due to the fact that the *Encoder* of dual-encoding architecture consumes more power than the corresponding module of a single-encoding architecture. Nevertheless, the dual-encoding architecture obtains almost the same power saving as the single-encoding architecture while introducing significant less area overhead.

5. CONCLUSION AND DISCUSSION

In this article, we proposed a new bipartition dual-encoding architecture for low-power pipelined circuits. The proposed scheme exploits a bipartition approach as well as encoding techniques in a pipeline stage to reduce power dissipation not only of combinational logic blocks but also of the pipeline registers. We bipartition a given circuit described by PLA into two sub-PLAs such that the number of different outputs of both PLAs are minimal. We then encode the outputs of both sub-PLAs to minimize the Hamming distance of register values with high transition probability.

The differences between precomputation and bipartition, bipartition single-encoding and bipartition dual-encoding architectures can be summarized as follows: compared to precomputation architecture, the precomputation approach only disables some of the input pins to reduce the switching activity of the combinational logic. However, the remainder input signals may also incur redundant switching activity in the entire combinational logic. Furthermore, precomputation does not account for the power dissipation of pipeline registers. Conversely, bipartition and bipartition single/dual-encoding architectures separate the combinational logic to ensure they will not influence each other. In addition, bipartition signal/dual-encoding architectures also take the power dissipation of pipeline registers into account by applying a codec structure, which significantly reduces power dissipation.

Our accurate transistor-level simulations demonstrate the practical impact of the partition and encoding approaches in lowering the power of pipelined circuits. Up to 92.7% and 67.9%, for register and overall power reduction, respectively, and 72.7% and 27.1% on average for pipeline registers and total power consumption, respectively, can be obtained by the bipartition dual-encoding architecture.

REFERENCES

Alidina, M., Monterio, J., Devadas, S., Devadas, S., Ghosh, A., and Papaefthymiou, M. 1994. Precomputation-based sequential logic optimization for low power. *IEEE Trans. VLSI Syst. 2*, 4 (Dec.), 426–436.

Benini, L. and Micheli, G. D. 1995. State assignment for low power dissipation. *IEEE J. Solid-State Circ. 30*, 3 (March), 258–268.

Chen, P.-H., Ruan, S.-J., Wu, K.-P., Hu, D.-X., Lai, F., and Tsai, K.-L. 2001. An Entropy-based algorithm to reduce area overhead for bipartition-codec architecture. In *Proceedings of the IEEE International Symposium on Circuits and Systems*. V-49–V-52.

Choi, I.-S. and Hwang, S.-Y. 1999. Circuit partition algorithm for low-power design under area constraint using simulated annealing. *IEE Proc. Circ. Dev. Syst. 146*, 1 (Feb.), 8–15.

Kapadia, H., Benini, L., and Micheli, G. D. 1999. Reducing switching activity on datapath buses with control-signal gating. *IEEE J. Solid-State Circ. 34*, 3 (March), 405–414.

Micheli, G. D. 1994. *Synthesis and optimization of digital circuits*. McGraw-Hill, New York, NY.

Moterio, J., Devadas, S., and Ghosh, A. 1993. Retiming sequential circuits for low power. In *Proceedings of the IEEE/ACM International Conference on Computer Aided Design*. 398–402.

Munch, M., Wurth, B., R.Mehra, Sproch, J., and Wehn, N. 2000. Automating RT-level operand isolatoin to minimize power consumption in datapaths. In *Proceedings of the Design, Automation and Test in Europe Conference and Exhibition*. 624–631.

Ruan, S.-J., Shang, R.-J., Lai, F., Chen, S.-J., and Huang, X.-J. 1999. A bipartition-codec architecture to reduce power in pipelined circuits. In *Proceedings of the IEEE/ACM International Conference on Computer Aided Design*. 84–89.

Ruan, S.-J., Shang, R.-J., Lai, F., and Tsai, K.-L. 2001. A bipartition-codec architecture to reduce power in pipelined circuits. *IEEE Trans. Comput.-Aided Des. 20*, 2, 343–348.

Schimpfle, C. V., Simon, S., and Nossek, J. A. 1997. Optimal placement of registers in data paths for low power design. In *Proceedings of the IEEE International Symposium on Circuits and Systems*. 2160–2163.

Tiwari, V., Malik, S., and Ashar, P. 1998. Guarded evaluation: Pushing power management to logic synthesis/design. *IEEE Trans. Comput.-Aided Des. 17*, 10 (Oct.), 1051–1060.

Ye, W. and Irwin, M. J. 1999. Power analysis of gated pipeline registers. In *Proceedings of the Twelfth Annual IEEE International ASIC/SOC Conference*. 281–285.

Received January 2003; revised August 2003; accepted March 2004

A Scheduling Algorithm for Optimization and Early Planning in High-Level Synthesis

SEDA OGRENCI MEMIK
Northwestern University
RYAN KASTNER
University of California, Santa Barbara
ELAHEH BOZORGZADEH
University of California, Irvine
and
MAJID SARRAFZADEH
University of California, Los Angeles

Complexities of applications implemented on embedded and programmable systems grow with the advances in capacities and capabilities of these systems. Mapping applications onto them manually is becoming a very tedious task. This draws attention to using high-level synthesis within design flows. Meanwhile, it is essential to provide a flexible formulation of optimization objectives as well as to perform efficient planning for various design objectives early on in the design flow. In this work, we address these issues in the context of data flow graph (DFG) scheduling, which is an essential element within the high-level synthesis flow. We present an algorithm that schedules a chain of operations with data dependencies among consecutive operations at a single step. This local problem is repeated to generate the schedule for the whole DFG. The local problem is formulated as a maximum weight noncrossing bipartite matching. We use a technique from the computational geometry domain to solve the matching problem. This technique provides a theoretical guarantee on the solution quality for scheduling a single chain of operations. Although still being local, this provides a relatively wider perspective on the global scheduling objectives. In our experiments we compared the latencies obtained using our algorithm with the optimal latencies given by the exact solution to the integer linear programming (ILP) formulation of the problem. In 9 out of 14 DFGs tested, our

At the time the research for this article was carried Out, S. O. Memik was affiliated with the University of California, Los Angeles, Los Angeles, CA.

Authors' addresses: S. O. Memik, Department of Electrical and Computer Engineering, Northwestern University, 2145 Sheridan Road, Evanston, IL 60208; email: seda@ece.northwestern.edu; R. Kastner, Department of Electrical and Computer Engineering, University of California, Santa Barbara, Engineering I, Room 4123, Santa Barbara, CA 93106; email: kastner@ece.ucsb.edu; E. Bozorgzadeh, Computer Sciences Department, University of California, Irvine, 408E Computer Science Building, Irvine, CA 92697-3425; email: eli@ics.uci.edu; M. Sarrafzadeh, Computer Science Department, University of Califonia, Los Angeles, 3532C Boelter Hall, Los Angeles, CA 90095-1596; email: majid@cs.ucla.edu.

algorithm found the optimal solution, while generating latencies comparable to the optimal solution in the remaining five benchmarks. The formulation of the objective function in our algorithm provides flexibility to incorporate different optimization goals. We present examples of how to exploit the versatility of our algorithm with specific examples of objective functions and experimental results on the ability of our algorithm to capture these objectives efficiently in the final schedules.

Categories and Subject Descriptors: B.5.2 [**Register-Transfer-Level Implementation**]: Design AIDS—*Automatic synthesis*; J.6 [**Computer-Aided Engineering**]: *Computer-aided design*

General Terms: Design, Algorithms

Additional Key Words and Phrases: Scheduling, high-level synthesis, data flow graph, bipartite matching

1. INTRODUCTION

Traditionally, translation of application descriptions into a synthesizable hardware description language (HDL) was a manual process. Then, designs specified with an HDL were mapped onto embedded or programmable systems by logic and physical synthesis tools. Due to the increasing complexity of the applications, mapping applications manually is becoming harder. Therefore, increasing the level of abstraction for designers and automating the mapping process is becoming more attractive. This alternative paradigm involves automatic compilation from high-level descriptions of applications, such as from a high-level programming language (e.g., C, C++). This approach offers designers a very convenient and familiar computational model. There are several proposed compilation flows from a high-level of abstraction to hardware [Wazlowski et al. 1993; Hammes et al. 1999; Gokhale et al. 2000; So et al. 2002; Haldar et al. 2001; Schreiber et al. 2002].

Figure 1 depicts an example flow for automatic mapping of applications onto various hardware platforms. The application described in a high-level programming language is processed by the compiler stage. The compiler generates an *intermediate representation* (IR) and performs several optimizations such as constant propagation, loop unrolling, and function inlining on this IR. While internal representations in different compilers take different forms and names, essentially they capture two basic pieces of information about an application: control flow and data dependency. A high-level synthesis stage follows the compiler stage and takes the optimized IR as input and generates the *register transfer level* (RTL) description of the design. Back-end tools perform logic synthesis and physical synthesis on this RTL description and create the bit-stream data to program the target system. In the most general form of this flow, feedback paths between major steps may exist to incorporate physical level information into high-level synthesis or hardware/synthesis-related information into compiler stage (denoted by dotted arcs in Figure 1). Feedback is employed in such flows to improve the interaction between different design phases and refine the quality of the solution generated at each stage. However, feedback can also cause problems in convergence and design closure. In order to create a more coherent design flow, planning early on can be a better alternative. This can be achieved by planning of design objectives or through use of correct by construction methodology.

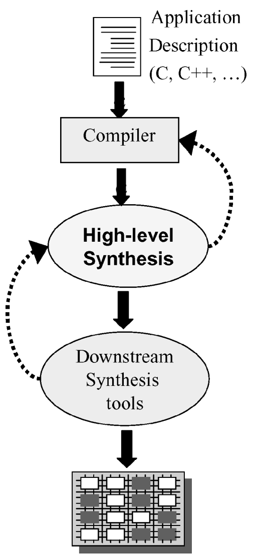


Fig. 1. A representative flow for automatic mapping of applications from high-level descriptions to a programmable system.

In this work, we present a scheduling algorithm for data flow graphs (DFGs), which is an integral element within the high-level synthesis stage of this automated flow. Our method "simultaneously" assigns a set of operations within the input DFG to control steps. (Note that scheduling several operations simultaneously does not refer to assigning them to the same control step. By scheduling them simultaneously we mean to generate a scheduling decision for a collection of nodes at once.) Each set of nodes selected to be considered together constitutes an ordered set. There exists a direct data dependency between every pair of consecutive nodes in each ordered collection of operations. We refer to these sets of operations as *paths*, since they constitute a topological path in the DFG. In the proposed algorithm, the scheduling problem for each individual set of nodes is formulated as maximum-weight noncrossing bipartite matching. This local problem in turn is solved optimally by converting it to the *max-weighted k-chain problem* [Atallah and Kosaraju 1989]. The bipartite matching basically provides the assignment of operations to control steps. DFGs impose a data dependency constraint on the scheduling problem. This is reflected in our algorithm by the noncrossing property of the matching solution. Furthermore, the matching is weighted and the objective is to produce a matching with maximum edge weight total. The particular objective function of the actual scheduling problem is embodied within the maximum-weight objective of the matching. We will elaborate on the specifics as we discuss our algorithm in detail. At this point, however, it is appropriate to comment on the impact of these properties on the global behavior of our scheduling algorithm.

First, our algorithm assigns several nodes along a path to control steps at once. This local assignment is realized by solving the matching between operations and control steps optimally. Each operation-control step matching within this solution is associated with a corresponding weight/gain. The local solution is optimal for the given set of operations in the sense that the summation of the weights of the matching generated for those operations is maximum. This distinguishes our algorithm from other heuristics such as list scheduling [Parker et al. 1986; McFarland et al. 1988; Pangrle and Gajski 1987], force-directed scheduling [Paulin and Knight 1987; Cloutier and Thomas 1990], etc., which generally make a scheduling decision about a single operation at a time. In our algorithm we generate a solution for a collection of operations while maximizing the scheduling objective for this set of operations. While the quality guarantee remains local, this helps to provide a good solution at each step for the operations along each path at hand. Furthermore, by manipulating the weight function associated with the matching between operations and control steps, a wide variety of objectives can be combined within a solution. Hence, our algorithm provides a flexible way of defining and changing our scheduling objective function. In this article, we discuss two specific cases in more detail: efficient utilization of optimized cores embedded in the target architecture and early planning and distribution of time slack within a DFG during scheduling.

The rest of the article is organized as follows. Section 2 states the scheduling problem and defines the objective and the constraints. We give a brief overview of existing scheduling heuristics in Section 2.2. Our algorithm is described in Section 2.3. In Section 3 we present how our algorithm can be applied to target two particular objective functions for schedules. First, we discuss using our scheduler for hybrid target architectures, aiming to optimize the utilization of embedded cores within the target architecture. Next, we present how our flexible scheduling cost function can be used to target efficient distribution and management of time slack within a schedule. We present experimental results for these two problem instances. We discuss our conclusions and future work in Section 4.

2. SCHEDULING OF DATA FLOW GRAPHS

In this section we formulate our problem and state the constraints on the problem. We also define our objective function. Next, we present our scheduling algorithm.

2.1 Problem Formulation

Given a data flow graph (DFG),[1] the scheduling problem is to assign each operation in the DFG to a control step under certain constraints. Any assignment that is feasible under these constraints is a valid schedule. Out of possible valid schedules the goal is to find one that optimizes a given objective function. An immediate objective function for any scheduling algorithm is the length of the schedule or the latency. In addition, depending on the specific context in which the scheduler is used, other components are incorporated into the objective function. Objectives such as power [Musoll and Cortadella 1995; Monteiro

[1]A data flow graph is basically a directed acyclic graph (DAG).

et al. 1996; Shiue and Chakrabarti 2000] and register usage [Wong et al. 2002] have been incorporated into scheduling algorithms in the past. In this work we will introduce two other objective functions and show how the cost function of our algorithm can be easily adjusted to include either of those. We will discuss these objective functions in Section 3. The objective function is maximized under a set of constraints. For our scheduling problem the following constraints are given:

—For each operation a start time must be defined.
—Data dependencies imposed by the DFG must be obeyed. Let, op_i and op_j be two operations in the DFG. In addition, let op_i be the parent of op_j. This means that op_j has a data dependency on op_i. Then, the control step in which op_j starts must be later than the finish time of op_i.
—At any control step, the number of active operations of any type must be less than or equal to the number of available resources of that type.

2.2 Scheduling Algorithms Overview

Most of the practical formulations of the scheduling problem are NP-complete. These instances contain combinations of dependency, timing, and resource constraints. Several efficient heuristics have been proposed in the literature for these problem instances. Two of the most widely used approaches are *list scheduling* and *force-directed scheduling*. List scheduling maintains an ordered list of operations that can potentially be scheduled at a control step with no violation of data dependency. Considering one control step at a time, operations are selected from this ordered list one by one according to some priority function and scheduled at the control step under consideration. There exist a variety of realizations of this approach [McFarland et al. 1988; Thomas et al. 1990; McFarland 1986; Parker et al. 1986; Pangrle and Gajski 1987; Kramer and Rosenstiel 1990]. In force-directed scheduling [Paulin and Knight 1989] the goal is to create a balanced distribution of operations among control steps. Using the mobility of each operation to define possible intervals of execution, the potential *demand* for each control step is determined. The operation-to-control step assignment, which will contribute toward the most homogeneous distribution is accepted at each step. This approach has been incorporated into high-level synthesis systems as well [Paulin and Knight 1987; Cloutier and Thomas 1990]. Another type of scheduling method is referred to as *path-based scheduling* in the literature [Camposano 1991]. This method has been proposed for scheduling control flow graphs. Paths of execution within the control/data flow are handled individually. Each such possible path is scheduled independently in an optimal fashion. Then the final schedule is constructed by imposing the resource constraints and overlapping the path's schedules accordingly. Other popular techniques for scheduling with control flow are trace scheduling from microcode compaction [Fisher 1981] and percolation scheduling [Potasman et al. 1990]. While most of the above-mentioned scheduling heuristics produce a scheduling decision for one operation at a time, our algorithm generates an assignment between multiple operations and control steps at once. The particular set of operations to be scheduled at each step constitute a chain of data dependency. This may be also called a *path spanning* through the DFG. In path-based scheduling algorithms mentioned earlier such as [Camposano 1991], any sequence of operations that can possibly be executed conforming to the control flow is a path. Those operations do not need to create a chain with direct data dependencies. Therefore the path-based scheduling algorithms consider an exponential number of possible execution paths and combine their schedules eventually. In our scheme the definition of a path is restricted to chains of operations with data dependencies among them. The number of such paths extracted from an input DFG is kept small, just enough to have considered all operations within the DFG. In fact, we can find such a set of paths for a DFG in polynomial time.

Our technique to perform the actual assignment of operations to control steps is based on weighted noncrossing bipartite matching. This approach is fundamentally different from list scheduling and force-directed scheduling, and path-based scheduling in nature. Moreover, at the local level, we can provide theoretical guarantees on the quality of the operation-control step assignment for individual operation chains. An algorithm proposed by Timmer and Jess [1995] uses bipartite graph matching for scheduling DFGs. In their work, a bipartite matching was performed between operations in the DFG and control steps without considering dependencies and the result was pruned with a heuristic in order to comply with precedence constraints. In our approach we show how to *optimally* solve the matching problem between a set of operations along a path and control steps *while satisfying* precedence constraints within the path. Hence, we have integrated the two problems handled separately by Timmer and Jess [1995]. Also, the method proposed by Timmer and Jess does not incorporate any objective function into the bipartite matching stage. In our method, we are additionally maximizing an objective function by generating a *maximum weight* matching. We will present a detailed analysis of our algorithm in the next section.

2.3 Our Scheduling Algorithm

Our scheduling algorithm consists of two main tasks. First, selecting a chain of operations to schedule together. Next, scheduling this set of operations by generating a noncrossing maximum-weight bipartite matching. The input to our algorithm is a data flow graph, and resource constraints for each resource type. In the following we will present details of the two main tasks within our algorithm.

2.3.1 *Selection of Operation Chains.* At each step we extract a chain of operations from the input DFG. We start with extracting the longest path from the DFG. After one set of operations are scheduled those operations are removed from the DFG and the next longest path within the remaining graph is found. In this manner paths are selected in decreasing order of path delays. While doing that we retain information regarding the latencies of the scheduled operations. In other words, for nodes already scheduled in the DFGs their delay is added on top of the expected delays of remaining paths to preserve the ordering of criticality among different paths. Finding the longest path in a directed acyclic

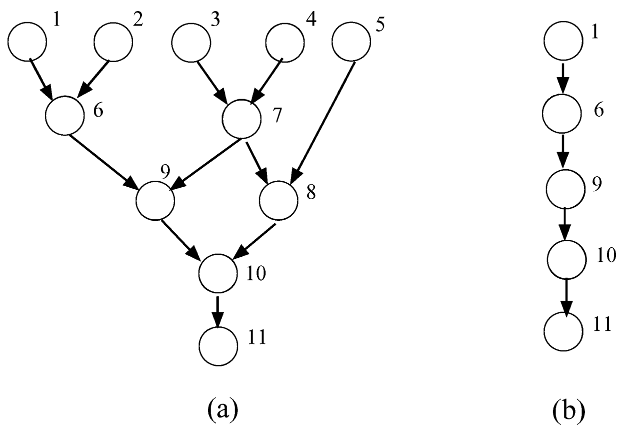


Fig. 2. (a) Example data flow graph. (b) A sample path extracted from the data flow graph.

graph (DAG)—as a graph type, a DFG is basically a DAG—can be done in polynomial time, in $O(V + E)$ time (V is number of nodes and E is number of edges in the DAG). The first chain of operations we process are those along the critical path of the DFG.

As the algorithm progresses the schedule of each new set of operations is constrained by the existing partial schedule of the DFG so far. The task of scheduling a given chain of operations at each scheduling step is tackled with a geometric approach.

2.3.2 *Scheduling Using Maximum-Weight Noncrossing Bipartite Matching.* The input to the local problem, that is, scheduling of a path, is a chain of operations. Consider the example DFG shown in Figure 2(a). Figure 2(b) illustrates a chain extracted from this DFG.

We visualize the problem of scheduling the operations along this chain as a bipartite matching between operations and control steps. A bipartite graph depicting this formulation for the chain in our example is shown in Figure 3(a). This bipartite graph consists of nodes corresponding to operations and control steps. An edge between an *operation-node* and a *control step-node* represents the possibility of assigning that operation to that control step. Constraints of the scheduling problem decide whether it is feasible to have an edge between a certain operation and a control step. For instance, at some point during the execution of our algorithm, at a certain control step the number of operations scheduled might have reached the number of available resources of some type. As we build the bipartite graph for the next set of operations to match to control steps, there cannot be an edge between any operation demanding that resource type and this control step. For any two consecutive operations along the path, the earliest possible matching between the child any control step must be later than the earliest possible matching between the parent and any control step. The latencies of individual operations may be single cycle or multiple cycles. When deciding the latest cycle at which a predecessor can finish we take this into account for multicycle operations. For instance, consider the operations

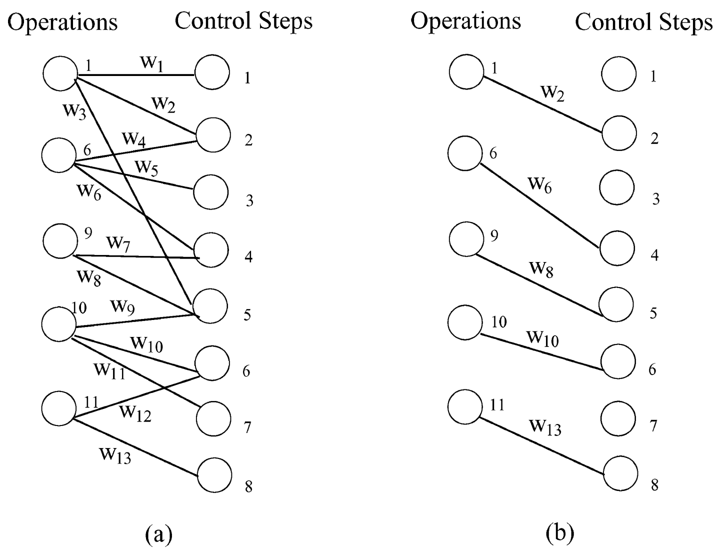


Fig. 3. (a) Bipartite graph representing the matching between operations in a path and control steps. (b) A noncrossing bipartite matching between operations and control steps.

Op_6 and Op_9. The earliest possible matching for the parent operation is to control step 2. Therefore, there cannot be any edge between the child operation, Op_9, and any control step earlier or equal to control step 2. Also, as operations from extracted paths are scheduled, their start and finish times restrict the intervals of control steps within which their predecessors and successors need to be scheduled. Our matching is also weighted, that is, each edge in the bipartite formulation is assigned a weight. In fact, for each path under consideration our goal is to find a matching such that the sum of the edge weights in the matching is maximum. The edge weights are used to formulate the objective function of the schedule; hence maximizing the sum of the weights in turn maximizes our objective function.

A mathematical function F needs to be defined to compute the weights. F is constructed according to what we want to achieve in our schedule. In different cases and applications different objectives can be more relevant or important. A traditional objective for scheduling is minimizing latency. However, a versatile and flexible scheduling algorithm should be able to consider other objectives depending on the particular context. Our approach in constructing F is to combine different optimization objectives in a weighted sum. User-defined coefficients for each term of F maintains the balance between different objectives. In the next section we will provide specific examples of objectives that can be incorporated into our cost functions.

A valid schedule for a path is a noncrossing matching between operations and control steps. This is a different problem than the general bipartite matching. Techniques, such as max-flow [Cormen et al. 1990], to find a bipartite matching do not guarantee yielding a noncrossing matching. A noncrossing bipartite matching means that in the matching solution no edges are allowed to cross.

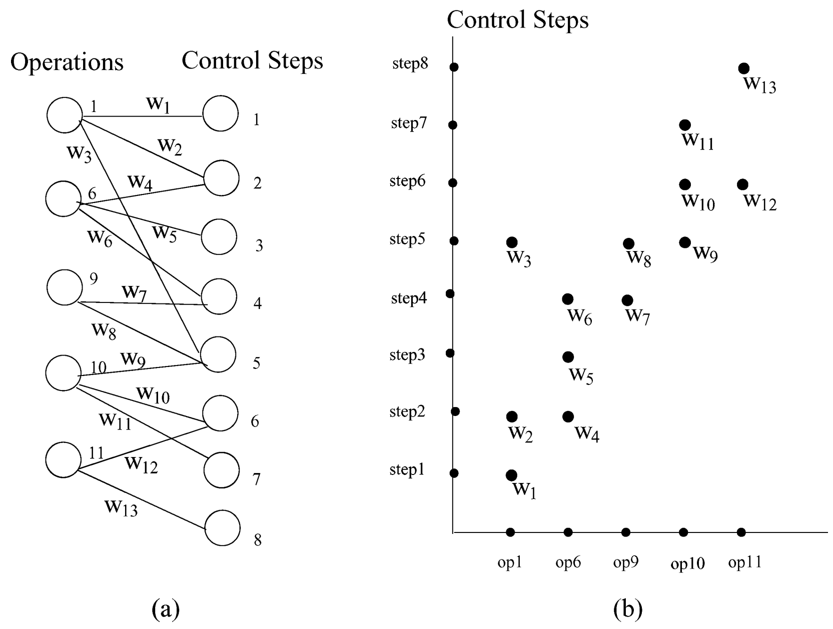


Fig. 4. (a) Weighted bipartite formulation for scheduling a path. (b) The geometric formulation of scheduling.

This actually corresponds to the data dependency constraint. Since along each path consecutive operations have data dependency, their matching to control steps must be noncrossing. A possible noncrossing matching for the example path from Figure 2(b) is shown in Figure 3(b).

Now the question is how to find a noncrossing matching with maximum weight total out of all possible noncrossing matchings. The bipartite matching formulation is a conceptual aide to visualize our problem. To find the actual solution, we will transform our problem into geometric domain and use the dominance concept in computational geometry [Lee 1996; Atallah and Kosaraju 1989] to solve our scheduling problem. To do this we first create a point in the x-y plane for each possible matching between operations and control steps. Figure 4 shows the original bipartite graph and the corresponding set of points in the x-y plane for our example path. Operations are placed along the x-axis and the control steps are placed along the y-axis. A possible matching between an operation Op_i and control step c is represented by a point in the plane with coordinates (x, y). To create the point set for a path of length k (k is equal to 5 for our example, since there are 5 operations along the path), we enumerate the operations with indices starting from 1 to k, following the topological order of the operations along the path. Then, the coordinates of each point representing the edge between Op_i and control step c are defined as follows:

$$x(Op_i) = index(Op_i) \quad \text{and}$$
$$y(Op_i) = c.$$

In our example index(Op_1) = 1, index(Op_6) = 2, and so on. The weights associated with edges in the bipartite graph are attached to the points on the plane as weights.

The following definitions will provide the background and the necessary terminology for understanding our method to perform the scheduling.

Definition 1. On the two-dimensional plane, point P *dominates* point Q iff $x(P) > x(Q) AND y(P) > y(Q)$.

Definition 2. A set of k points $(P_1, P_2, \dots, P_k)$ in the x-y plane, where P_i dominates P_{i-1} is called a *k-chain*.

Definition 3. Given a set of points in the x-y plane, among all chains of length k that exist within this set, the k-chain with the maximum total of weights is called the *maximum weighted k-chain*. The weights are those attached to each point in our formulation.

Having created a set of points in the plane as explained above, the schedule for the path is generated by finding the *maximum weighted k-chain* within this point set. k is equal to the number of operations on the path that is being scheduled. By finding a chain of length k, that is, selecting k points from the plane, we will have found a matching between each operation and a control step. The property described in Definition 1 ensures that the matching found complies with dependencies among operations. Also we will ensure that our method finds a chain of length k and does not return any chain of shorter length, which would leave some operations unscheduled. Once we guarantee finding a chain of length k, combined with the dominance property from Definition 1, each point in the resulting k-chain must correspond to a matching for a distinct operation. Therefore, this k-chain corresponds to the schedule of the path. Assuming that we find a chain of length k, no two points can have the same x-coordinate, since the k-chain would not possess the dominance property from Definition 1 in that case. Hence we guarantee that each operation is included in the solution with a valid matching. We will explain how we guarantee finding a chain of length k every time after we introduce the method of finding the *maximum-weighted chain*. For illustrative purposes, a possible k-chain from the point set of our example and the corresponding partial schedule is shown in Figure 5.

Atallah and Kosaraju [1989] proposed an optimal $O(n \log n)$ (n is the number of points in the plane) algorithm for finding the maximum-weight k-chain. We refer the readers to Atallah and Kosaraju [1989] for the proof of optimality. If for each point P in the plane the weight $w(P) = 1$, then this algorithm returns the longest possible chain, which naturally corresponds to the maximum sum of weights. However, when arbitrary weights are assigned to the points in the plane, this algorithm yields the maximum weighted chain, but not necessarily of length k. As explained earlier, we need to guarantee a chain of length k and depending on our weight function F the weights can take arbitrary values. We propose an adjustment to the weights, such that the algorithm proposed by Atallah and Kosaraju [1989] can be adapted to our problem.

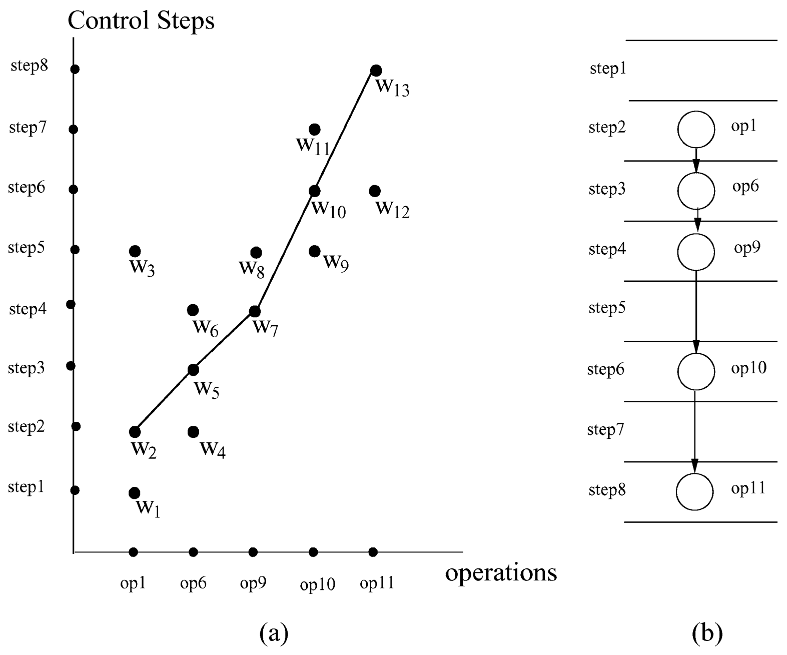


Fig. 5. (a) A hypothetical matching found using a maximum-weighted k-chain. (b) Corresponding partial schedule.

THEOREM 2.1. *Let the weight $w(P_i)$ of each point P_i in the plane have arbitrary values. If each weight $w(P_i)$ is replaced with*

$$w(P_i)' = 1 + \frac{w(P_i)}{\left(\sum_{i=1}^{n} w(P_i)\right) + 1}$$

then the maximum weighted chain will be of length k, if any chain of length k exists in the point set.

PROOF. Given a set of n points at k different x-coordinates (because we have k operations along the path to be scheduled), assume there exists a maximum weighted chain of length $k - 1$. Then, the sum of weights on this chain would be

$$(k-1) \cdot 1 + \sum_{i=1}^{k-1} \frac{w(P_i)}{\left(\sum_{i=1}^{n} w(P_i)\right) + 1}.$$

The second term in the above sum is the sum of weights for $k - 1$ points divided by the total sum of weights plus 1. This term is always less than 1. Therefore,

$$k - 1 + \sum_{i=1}^{k-1} \frac{w(P_i)}{\left(\sum_{i=1}^{n} w(P_i)\right) + 1} < k.$$

On the other hand, if we take any k points (strictly one from each x-coordinate obeying dominance condition; assuming at least one chain of length k exists, we should be able to do that) the sum of the weights of these k, points will be

$$k \cdot 1 + \sum_{i=1}^{k} \frac{w(P_i)}{\left(\sum_{i=1}^{n} w(P_i)\right) + 1},$$

which is definitely larger than the sum of $k-1$ points. This contradicts the initial assumption of having a $(k-1)$-chain with a maximum-weight sum. Hence, the weight sum of any chain of length k will be larger than any chain of shorter length with this weight assignment. (The case for chains shorter than $k-1$ follows same arguments.) □

COROLLARY 2.1. *For any two weights w_1 and w_2, if $w_1 > w_2$, then $w_{1'} > w_{2'}$. This property ensures that if a k-chain found with adjusted weights is a maximum-weighted chain, then by converting back to the original weights we can show that the original weights would yield the same maximum-weight k-chain.*

So far we have explained how to guarantee finding a maximum-weighted k-chain if at least one chain of length k exists in the point set. To guarantee the latter, that is, that there is at least one k-chain in the point set, for each operation along the current path there needs to be at least one feasible matching control step. In other words, there must be at least one point with each x-coordinate from 1 to k. At each scheduling step, we check the point set generated for each path of operations to verify that this is true. If not, we perform a correction pass. This actually corresponds to situations when the schedules of different paths need to be combined. Note that in our algorithm once a path is selected all operations along the path are scheduled regardless of whether any of their predecessors are scheduled. Sometimes, the parent of an already scheduled operation is included in a path that is extracted from the DFG later in the execution of the algorithm. It can be the case that at some point due to resource and/or dependency constraints we cannot find any possible feasible matching between control steps and the parent operation that appeared later in the scheduling process. In these cases we restore the feasibility of the matching, that is, we ensure finding at least one chain of length k in the point set by inserting extra control steps into the schedule. We do this according to the following rules. Assume at some point we are about to schedule a path that contains an operation which has some already scheduled successor(s). If the earliest possible start for the predecessor operation is later than the earliest scheduled successor, then we insert extra control steps right before the earliest scheduled successor and push all operations starting at or later than this step by the delay of the predecessor operation. If the earliest possible start of the predecessor operation is earlier or at the same time as the earliest scheduled successor operations (probably the predecessor operation has large delay, such that its result is not ready for the earliest scheduled successor although it can start before the earliest scheduled successor), then we need to determine the number of extra control steps to be inserted as follows. If the earliest possible start time of the predecessor operation is equal to the start of successor, then we insert as many control steps as the delay of the predecessor operation. If the

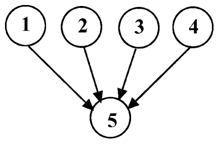


Fig. 6. A sample DFG where $O(V)$ correction passes will be required.

predecessor operation can start earlier than the earliest scheduled successor but cannot finish on time, then we push the schedule by the overlapping amount of control steps by inserting extra control steps.

In the following we will make a quick analysis of the worst-case behavior expected from our algorithm in terms of number of correction passes that have to be performed. Consider a DFG as shown in Figure 6. Assume that all operations are of same type and also assume that there is only one resource available. Finally, assume that all operations take a single control step. Given these conditions, our algorithm would first schedule a path such as $(Op_1{-}Op_5)$ in the first and second control steps. Then, the algorithm encounters Op_2 (plus the delay of Op_5) as the second path. At that point we will need the correction pass and push operation Op_5 by one control step to accomodate time for Op_2. Similarly, in the consequent steps of the algorithm, a correction pass will be required before scheduling Op_3 and Op_4. This shows that we might need corrections proportional to V, which is the number of operations in the DFG. Note that, in each step, the scheduling decision of at least one node (the node with no parent) will be finalized. Therefore, we will not need more than V correction passes. Consequently, in the worst case our algorithm will require $O(V)$ correction steps passes.

The need for correction passes is correlated with three factors. First, it is related to the DFG topology. For DFGs with low connectivity and with more independent paths spanning throughout the DFG, possibly there will be fewer conflicts. Second, the resource constraint will impact the frequency of occurence of conflicts. Consider the earlier example of Figure 6. If there were four resources available, there would be no need for correction passes. If there were three resources, there would be a single correction pass. For more stringent resource constraints, the conflicts during scheduling will increase. Finally, the selection of paths is important. We try to minimize possible conflicts by scheduling the longest paths first. By doing this, operations on long paths would be more likely to be scheduled much later than their yet unscheduled predecessors in shorter paths.

Although our algorithm is locally optimal, it might not yield a globally optimal schedule. This lack of optimality is related to the correction passes. Although we schedule a given path optimally under the given conditions, it does not mean that the final global schedule will require this path to be scheduled within the shortest time. In fact, correction passes try to merge optimally scheduled paths into nonoptimal paths that will be required by the optimal global schedule. However, due to the selection order of the paths and our local decision-making mechanism, we might not be able to merge into the globally optimal schedule.

```
geom_Scheduling(DFG(V,E))
1 while there are unscheduled operations in the DFG
2         Select the most critical (partial) path
3         Generate the point set in the X-Y plane
4         Check for feasibility of point set (insert extra control steps if needed)
5         Assign weight_i = F(p_i) to each point p_i
6         Apply max_weighted_k_chain()
7         Update the DFG
8 end while
```

Fig. 7. The overall scheduling algorithm.

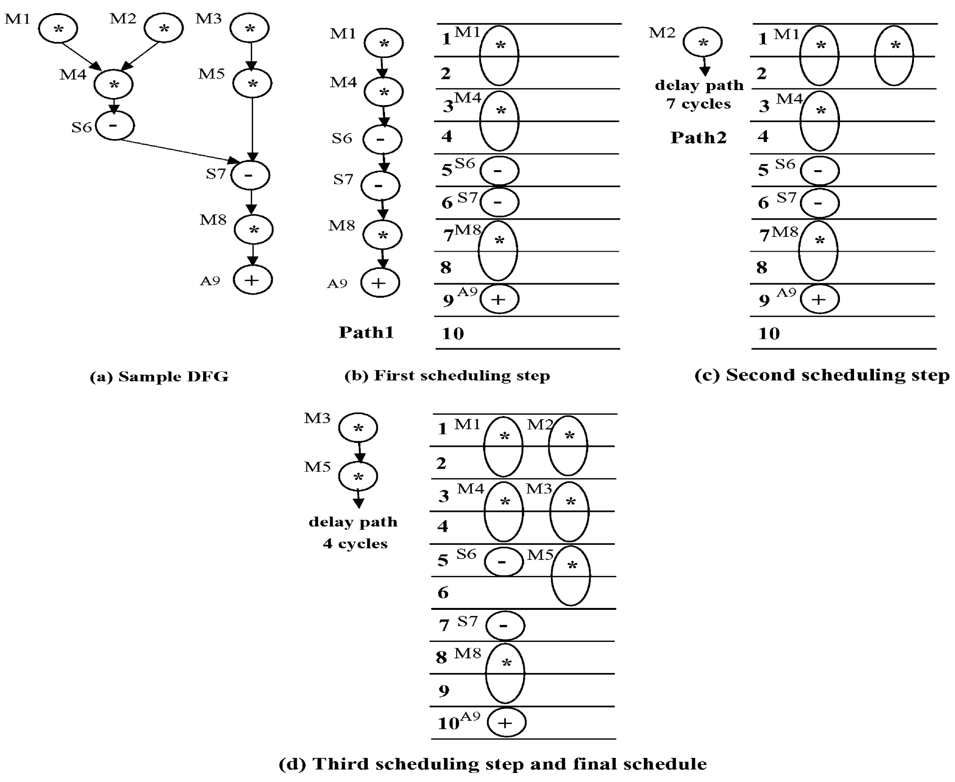

M1
M2
M3
M4
M5
S6
S7
M8
A9
(a) Sample DFG
Path1
(b) First scheduling step
delay path
7 cycles
Path2
(c) Second scheduling step
delay path
4 cycles
(d) Third scheduling step and final schedule

Fig. 8. Illustration of the algorithm with a sample DFG.

Combining all the steps explained above, the overall scheduling algorithm is summarized in Figure 7.

In Figure 8 we illustrate the execution of the algorithm on a sample DFG. We assume that one ALU to execute additions and subtrations and two multipliers is available. In addition, the latency of the ALU is a single cycle and the latencies of the multipliers are two cycles. In this case the scheduling algorithm executes in three steps. In the first step, the chain consisting of M1, M4, S6, S7, M8, A9 is scheduled. For this example we assume that weights of matchings are adjusted such that the earliest schedule step is preferred for each operation. In the second step, the operation M2 is scheduled. M2 is chosen next considering that M2 combined with the path following M2, which consists of already scheduled operations, has the next longest delay. In the third and last step, the chain consisting of operations M3, M5 is scheduled. However, at this step the initial bipartite graph created for this chain is infeasible. This is due to the fact that the earliest possible matching between operation M5 and a clock step with a free multiplier is later than the step at which the successor of M5, which is S7, is scheduled. Therefore, at this point a correction step is applied. S7 and all transitive successors of S7 are shifted by a cycle. After this correction, step M3 and M5 can be successfully scheduled. The final schedule is shown in Figure 8(d).

3. TUNING THE MAXIMUM WEIGHTED MATCHING FOR SCHEDULING OBJECTIVES

We can use the maximum-weighted matching procedure to optimize different objectives within the schedule. Any possible matching between an operation and a control step would contribute to the quality of the certain feature(s) that we aim to embody in the final schedule. Different assignments of operations to control steps can result in different amounts of resource requirements, interconnect structure, switching activity, operation slack, etc., in the final synthesized design. In this article we present how we can utilize the flexibility of the weight assignment to pursue two specific objectives: utilization of embedded fixed cores within an embedded system or within the programmable fabric of a reconfigurable device, and early planning and distribution of slack within a schedule.

An immediate objective for our scheduling algorithm is to minimize latency. In order to optimize for this objective only, we need a function that assigns monotonically decreasing values to weights. We need such a distribution among the weights assigned to all feasible matchings between an operation and all candidate control steps. In other words, for each operation the weight of matching it with a certain feasible control step c must be larger than matching the same operation with any *later* feasible control step. A possible function to create these weights could be as follows:

$$F = \kappa - c.$$

Here, κ is some constant and c is the index of the control step. As we go further in time axis, the value of F will be monotonically decreasing.

3.1 Embedded Core Utilization Objective

A possible objective in high-level synthesis can arise due to the specific architectural features of the target system. For instance, looking back at the evolution of programmable systems, configurability was first available in standalone chips. These devices possess 100 % programmability. Currently, reconfigurable fabric is not only considered to be confined to standalone chips, but also is part of hybrid systems such as system-on-chip (SoC) solutions. While one trend is toward embedding reconfigurable cores into SoCs with processors, DSPs, etc. [Hauck et al. 1997; Actel ; Altera Corp. ; Chameleon Systems], another direction of new architectures considers integration of optimized cores and hardwired blocks with reconfigurable fabric. The main goal here is to utilize the optimized blocks to improve the system performance. Such programmable devices are targeted for a class of applications, such as DSP [Xilinx, Inc.] networking, or data communications [Lucent Technologies]. Embedded fixed blocks are tailored for the critical operations common to the application class. In essence, the flexible programmable logic is supported with the high-density, high-performance cores. This can be applied at various levels, such as the functional block level [Lucent Technologies] or the level of basic arithmetic operations, for example, multipliers [Xilinx, Inc.].

In the context of reconfigurable systems, there have been efforts to create compilation frameworks where templates for frequently occurring operations or operation clusters are extracted and mapped onto specialized cores. Such a framework is described by Kastner et al. [2001]. In a synthesis environment, where such templates are recognized for a given application during compilation, it is crucial to utilize these optimized modules during the actual synthesis of the datapath at the high-level synthesis stage.

For mapping designs on such special architectures, there is a need for synthesis tools that are aware of the features of the underlying hardware resources. The customized cores certainly improve the application's running time since they are superior in delay to their counterparts implemented with reconfigurable logic. A similar argument is valid for the power consumption. Finally, those blocks will lessen the reconfiguration overhead for the overall design. The power of the context-based reconfigurable architectures lies in the efficient utilization of the fixed cores within the system. By customizing our weight function F accordingly, we can make our scheduler aware of the customization of the target architecture. As a result, as we schedule DFGs we can exploit the optimized embedded cores without causing their limited availability become a bottleneck. Functional units for any desired operation type can be instantiated using reconfigurable logic. For operations that cannot be performed by the embedded cores, this is a necessity. For other operations, this can be done in order to exploit parallelism in the schedule. However, as mentioned earlier, the available blocks are highly preferred for those operations. It is the task of the scheduler to do the tradeoff in such situations.

When considering two different feasible control steps for a matching with an operation, the control step with a free customized block would be preferred over the other control step at which no customized embedded block is free. There still can be a feasible matching between the operation and the later step, assuming a reconfigurable module is available or can be instantiated. Nevertheless, the weight assigned to the first feasible matching should be made larger in order to make the algorithm aware of the resource preferences. Also, the value of the weight for the matching of the latter step can reflect the willingness of the synthesis to instantiate new reconfigurable modules. If this matching is associated with a very small value, the tendency would be to avoid instantiation of further reconfigurable modules, possibly in order to control the reconfiguration

Table I. Origins of DFGs from MediaBench

Benchmark	C File	Description
adpcm	adpcm.c	ADPCM to/from 16-bit PCM
epic	convolve.c	2D image convolution
rasta	fft.c	Fast Fourier Transform
mpeg2	getblk.c	DCT Block Decoding
jpeg	jdmerge.c	Color Conversion

overhead. We can introduce these objectives as additional terms within the weight function F. In this case F can take the following form:

$$F = \kappa - \alpha \times c + \beta \times \mathit{customized\_block\_preference} - \gamma \times \mathit{reconfiguration\_overhead\_preference}.$$

We can formulate architecture-related preferences as Boolean variables taking two values, 0 or 1. α, β, and γ are user-defined constants. We have adjusted their values for our algorithm experimentally.

3.1.1 *Results for Embedded Core Utilization Objective.* We have used DFGs extracted from representative functions of C programs within MediaBench multimedia benchmark suite [Lee et al. 1997] and also some additional representative DSP functions such as EWF, FIR, and ARF. Table I shows the files from MediaBench suite, from which input DFGs were generated. The corresponding applications containing these files are given as well. Two or more DFGs were extracted from different procedures within these C Files. Those are indicated as DFG0, DFG1, and DFG2 within each application. These DFGs were mainly selected due to the fact that they were representative of the corresponding applications in terms of size, topology, and operation variety. We tried to select the largest possible DFGs out of those extracted from the Mediabench applications. Each of these DFGs correspond to a basic block[2] in the application codes. Many of those basic blocks tend to be small in size. We tried to avoid such small basic blocks. In order to increase the input DFG sizes and the parallelism, entities containing multiple basic blocks (e.g., hyperblocks [Mahlke et al. 1992], superblocks [Hwu et al. 1993], and traces [Fisher 1981]) can be equivalently given to our scheduler as input. We have not attempted to create such formations at this point, since it is beyond the scope of this work.

Individual DFGs are scheduled with two different methods, as shown in Table III. For each application, a set of hardware resources are specified as given in Table II. Within the available set of resources, there can be multiple components with same functionality, but different delays. This represents the existence of optimized cores for certain operations. In Table III scheduling results for a number of selected DFGs are given. We compare the results of our algorithm against the results obtained from the linear programming solver, CPLEX. The scheduling problem has been described as a linear integer program for our problem instance. The objective function of the integer linear model tries

[2]A basic block is the same entity as in the compiler terminology, which refers to a straight line code segment with a single entry and a single exit point.

Table II. Resource Sets of Benchmark DFGs

Benchmark	Resource set
adpcm	DFG20:(ior, add, comp, mult-fast, mult) DFG36:(comp-fast, comp, add, mult-fast, mult)

convolve	DFG0: (comp, add/sub, mult-fast, mult, div-fast, div)
	DFG19:(add, mult-fast, mult)
	DFG98:(comp, add/sub, mult-fast, mult, div-fast, div)

fft	DFG18:(add, mult-fast, mult)
	DFG27:(add, mult-fast, mult, div-fast, div)

getblk	DFG42:(add, comp, AshiftR, mult-fast, mult)
	DFG91:(add, comp, AshiftR, LshiftL, mult-fast, mult)
	DFG165:(add, comp, AshiftR, mult-fast, mult)

jdmerge	DFG2: (add, AshiftR, mult-fast, mult) DFG21:(add, AshiftR, mult-fast, mult)
	DFG165:(add, comp, AshiftR, mult-fast, mult)
ewf	(add, mult-fast, mult)
arf	(add, add, mult-fast, mult, mult)
fir	(add, mult-fast, mult)

Table III. Scheduling Results in Terms of DFG Latencies, in Cycles

Benchmark (DFG)	ASAP	CPLEX	Our algorithm
adpcm			
DFG1	6	7	7
DFG2	6	6	6
convolve			
DFG1	12	16	20
DFG2	6	12	12
DFG3	17	18	19
fft			
DFG1	12	26	27
DFG2	19	22	24
getblk			
DFG1	14	14	14
DFG2	18	18	18
jdmerge			
DFG1	7	28	31
DFG2	8	28	28
ewf	17	28	28
arf	8	12	12
fir	7	12	12

to minimize the latency and the number of *slow blocks* used. A higher priority is given to latency minimization. Similarly in our algorithm, we try to minimize the latency, while trying to utilize the optimized blocks as well as possible. We actually perform binding of operation to resources simultaneously with scheduling. This enables us to handle resources of same type but with different delay characteristics. Through simultaneous binding we obtain operation delay

Table IV. Number of Operations Performed on the Optimized Block Versus Total Number of Operations of Matching Type

Benchmark (DFG)	CPLEX	Our algorithm
adpcm		
DFG1	1/1	1/1
DFG2	1/1	1/1
convolve		
DFG1	4/6	3/6
DFG2	2/3	2/3
DFG3	5/6	5/6
fft		
DFG1	6/9	6/9
DFG2	5/6	**6/6**
getblk		
DFG1	3/4	3/4
DFG2	4/4	3/4
jdmerge		
DFG1	5/8	5/8
DFG3	4/5	3/5
ewf	3/8	3/8
arf	10/16	8/16
fir	7/11	7/11

information based on the particular resource executing the operation. CPLEX provides us an optimal solution for the given objective function. Therefore, we are comparing our results to those generated by the exact solver. For comparison, the ASAP scheduling latencies are also provided. In Table IV the utilization of high-performance components are presented. For each DFG the total number of operations assigned to optimized blocks is given versus the total number of operations of suitable type that can be performed by any available optimized block.

As depicted in Table III, our algorithm was able to produce latencies compatible with CPLEX results for most cases. In 9 out of 14 cases our algorithm was able to produce the optimal latency. Out of the remaining five cases, four were within 12% of the optimal value. In two of the suboptimal cases, convolve-DFG3 and fft-DFG1, our algorithm was able to utilize the optimized blocks as good as CPLEX results. In the case of fft-DFG2, the increase in latency resulted from a tradeoff aiming to increase usage of optimized blocks. Table IV shows that, for this particular DFG, our algorithm was able to outperform the optimized block utilization obtained from the CPLEX solution.

For 9 out of the 13 remaining cases our algorithm reached the same core utilization as the CPLEX solution. For six of those cases the latency produced by our scheduler was optimal. For the remaining three cases our scheduling solution was within 10% of the optimal value. In four cases our algorithm yielded lower core utilization. We observe that in three out of those four cases the latency produced by our scheduling algorithm was equal to the optimal latency. Hence, the core utilization, although lower than the solution produced by CPLEX, seems to have been sufficient to reach an optimal schedule. We report

Table V. Number of Nodes and Edges in Each Benchmark DFG and the Runtimes of Two Scheduling Methods

DFG	Size (nodes/edges)	CPLEX	Our algorithm
adpcm			
DFG1	17/19	26 s	4 ms
DFG2	21/18	20 s	9 ms
convolve			
DFG0	49/41	121 s	14 ms
DFG1	25/19	30 s	9 ms
DFG2	18/10	24 s	4 ms
fft			
DFG1	17/12	30 s	3 ms
DFG2	11/9	4.8 s	2 ms
getblk			
DFG1	33/29	34 s	12 ms
DFG2	40/30	42 s	13 ms
jdmerge			
DFG1	79/66	985 s	23 ms
DFG2	54/46	436 s	14 ms
ewf	34/47	13.67 s	25 ms
arf	28/30	700 s	14 ms
fir	21/20	8.8 s	4 ms

the sizes of the benchmark DFGs in terms of number of nodes and edges, and the compare the runtimes of the two schedulers in Table V.

3.2 Early Planning and Distribution of Slack

Another possible use of our flexible objective function is distribution of operation slack within a schedule. We define slack as the amount of extra delay an operation can tolerate without violating any dependency constraints. There can be various uses of this extra amount of allowed time per operation.

Depending on the available slack for an operation, resource selection, IP utilization, power management, clock tree construction, etc., can be different. To give a more specific example: from a single operation's point of view, slack on this operation can be exploited for slowing the module executing this operation or performing power shutdown for this module. Hence, early planning for this objective can have various uses and an impact on the later optimization stages.

We can incorporate this new objective into our weight function F in the following manner. Given a latency constraint λ, we evaluate the weight of a feasible matching for each operation considering the amount of slack that the operation can attain after this matching. Specifically, we aimed to distribute slack along paths homogenously at each step. For this purpose, we first determine the average slack each operation can have along a path. This is found by dividing the total slack along a path to the number of operations on the path. Then, while computing the weight for a certain assignment of an operation to a clock step, we determine the difference between the average slack the operation could attain and the amount it can attain if the current assignment were to take place. The latency constraint is necessary in this case, since the flexibility in finish times of operations needs to be bounded.

Table VI. Summary of DFG Properties, Latency, and Runtime Results

DFG	Optimal latency	Our latency	Runtime
ewf	28	28	10 ms
arf	18	20	7 ms
fir	16	16	6 ms

Table VII. Incorporating Operation Slack into Scheduling Objective

	ewf		arf		fir	
	W/O slack objective	With slack objective	W/O slack objective	With slack objective	W/O slack objective	With slack objective
Num. of ALU operations with nonzero slack	4	5	4	2	3	4
Total slack on ALU operations	14	16	9	8	7	9
Num. of MUL operations with nonzero slack	2	2	7	9	4	5
Total slack on MUL operations	3	3	23	26	6	8

3.2.1 *Results on Planning for Slack.* First, we have used a weight function that only considers to minimize latency. This corresponds to the Without (W/O) Slack Objective case. Alternatively, we have added the slack component to our weight function and used the latencies obtained in the first case as our λ. By doing this, we are able to compare two schedules fairly. Table VI presents the DFGs, the optimal latencies, the latencies obtained with our algorithm, and the runtimes of our algorithm. Table VII shows our results. For this experiment we present results for a subset of our original DFG collection. The particular DFGs selected were those with suitable topologies to reflect improvement in slack. Availability of slack in a schedule depends on the topology of the input DFG as much as on the scheduling method. Therefore, in some DFGs it was not possible to see any effect of slack planning due to their structure. For the selected DFGs, we have used two ALU resources and two multipliers. Each ALU has a delay of one clock cycle, and each multiplier has a delay of two clock cycles. The slack of operations is calculated as the difference between the control step when the result of an operation is ready and the control step at which the earliest scheduled successor of this operation demands the result. We report the number of operations that have nonzero slack values, that is, the rest of the operations in the scheduled DFG had slack values equal to zero or they were I/O operations for which we do not report slack. We only report the slack values of arithmetic operations (MUL and ADD). We also give the total sum of the slack values on each operation type. We present a breakdown of these two measurements for the two types of arithmetic operations in the DFG, that is, the ALU operations and multiplications (abbreviated as MUL in Table VII).

The results in TableVII show that with proper planning to distribute slack on arithmetic operations our algorithm could indeed transform available flexibility in a schedule into additional slack for a spefically targeted set of operations, arithmetic operations in this case. For the arf benchmark we observe a degradation in slack value for ALU operations. The reason is due to the assignment of priorities for the two operations types ALU and MUL. The sensitivity toward increase the slack for MUL operations was set higher for these experiments. Therefore, the gain of increasing the slack for MUL operations is evaluated to be larger than for ALU operations, leading to a greater tendency to allocate slack for MUL operations than for ALU operations. In the case of ewf, due to the DFG topology and the order in which operations were scheduled, MUL operations could not attain any slack whereas ALU operations could gain larger slack-using the slack objective. For the fir benchmark, slack objective affected the slack distribution for both operation types positively.

By allowing a larger number of operations to possess nonzero slack, further optimizations can be possible. For instance, by ensuring slack on an increased number of operations, we would have increased the tolerance of the schedule toward future uncertainties in operation delays. Those uncertainties can arise due to a mismatch between high-level delay estimations and actual operation delays after synthesis of functional modules and interconnect in the datapath.

Additional optimization steps can be incorporated into the synthesis flow following the scheduling in order to exploit the available slack. For instance, a slack-oriented binding methodology can take advantage of this planning by assigning operations with nonzero slack to the same resource. If operations that possess various amounts of slack were assigned to the same resource, then the minimum out of those slack values would determine the extra amount of time by which that resource could be made slower. Using this information, some resources can be replaced by their slower and more power- and/or area-efficient versions. We have investigated possible benefits of exploiting slack distribution in a schedule during binding where we used our scheduling algorithm integrated with a slack-driven binding technique. After distributing time slack to arithmetic operations, as described earlier, we performed binding while trying to group operations with large slack values to the same resource. By doing this, the delay constraint on that resource could be relaxed by the minimum amount of extra slack available on the operations assigned to the resource. Then this information was passed onto the logic synthesis tool as a delay constraint relaxation of the corresponding module. We have observed significant benefits through this relaxation in terms of final design quality. Results to this end are reported in Srivastava et al. [2003] in detail.

Similarly, even if some operations assigned to one resource posses nonzero slack while others have no extra slack, some power optimizations are still possible. In such cases, techniques such as dynamic voltage scaling and dynamic power shutdown can be used if the schedule has been planned for slack distribution early on.

Optimization for slack as described in this section can also be helpful for tasks preceding scheduling. By creating a feedback loop between high-level synthesis and the compilation stage, different compiler optimizations can be leveraged. One such optimization is template extraction and template matching during compilation. At early stages it can be beneficial to extract frequently occuring subgraphs as templates from the DFGs and replace them with optimized cores. Usage of such cores was discussed in Section 3.1. Especially for reconfigurable platforms, usage of preoptimized precharacterized modules can speed up the synthesis process and improve performance. However, information regarding scheduling and binding is not available at the compiler stage. Therefore, the decisions of the compiler to lump subgraphs into larger nodes corresponding to optimized cores might not be accurate enough. This is due to the fact that scheduling of nodes internal to such subgraphs can affect the latency of the overall DFG schedule. Therefore, scheduling and also distribution of time slack among nodes needs to be aware of any clustering of nodes. In turn, after scheduling, a more accurate assessment of the benefits of clustering can be done. Especially, it would not be desirable to leave time slack within a group of nodes which are going to be assigned as a subgraph onto a specialized core. Based on the slack distribution and the schedule information, the compiler can be informed about the quality of clustering decisions made earlier during compilation. If a certain template is not yieling any gain or if it is hurting the performance of the overall DFG schedule, the compiler can be advised to undo that template clustering or redo it in a different way based on the current schedule information.

4. CONCLUSIONS

In this article we presented an algorithm for scheduling data flow graphs. Our algorithm uses a geometric representation of the problem. We applied the maximum weight k-chain technique to generate schedules for paths extracted from the input DFG. This technique is essentially a realization of maximum-weight noncrossing bipartite matching in the geomteric domain. The maximum-weight k-chain method enables us to provide a theoretical guarantee on the quality of each local matching problem. By exploiting the weighted matching feature inherent in our technique, we are able to provide a flexible objective function for the scheduling problem. We manipulate the function that generates the weights for the matching in order to create suitable objective functions for different scheduling problems. Our experiments indicate good results in terms of the combined effort of minimizing latency and utilizing optimized cores available in a given system. We also demonstrated the use of the flexible objective function in managing operation slack within a schedule.

REFERENCES

ACTEL. Visit the Web site http://varicore.actel.com.

ALTERA CORP. Visit the Web site http://www.altera.com/products/prd-index.html.

ATALLAH, M. J. AND KOSARAJU, S. R. 1989. An efficient algorithm for maxdominance with applications. *Algorithmica 4*, 2, 221–236.

CAMPOSANO, R. 1991. Path-based scheduling for synthesis. *IEEE Trans. Comput.-Aided Des. 10*, 85–93.

CHAMELEON SYSTEMS. Visit the Web site www.chameleonsystems.com.

CLOUTIER, R. AND THOMAS, D. 1990. The combination of scheduling, allocation and mapping in a single algorithm. *Proceedings of the Design Automation Conference*.

CORMEN, T. H., LEISERSON, C. E., AND RIVEST, R. L. 1990. *Introduction to Algorithms*. MIT Press, Cambridge, MA.

Fisher, J. A. 1981. Trace scheduling: A technique for global microcode compaction. *IEEE Trans. Comput. 30*, 7, 478–490.

Gokhale, M. B., Stone, J. M., Arnold, J., and Kalinowski, M. 2000. Stream-oriented FPGA computing in the streams-c high level language. In *Proceedings of the IEEE Symposium on Field-Programmable Custom Computing Machines*.

Haldar, M., Nayak, A., Choudhary, A., and Banerjee, P. 2001. A system for synthesizing optimized FPGA hardware from matlab. In *Proceedings of the International Conference on Computer Aided Design*.

Hammes, J., Rinker, R., Bohm, W., Najjar, W., Draper, B., and Beveridge, R. 1999. Cameron: High-level language compilation for reconfigurable systems. In *Proceedings of the Conference on Parallel Architectures and Compilation Techniques*.

Hauck, S., Fry, T. W., Hosler, M. M., and Kao, J. P. 1997. The Chimaera reconfigurable functional unit. In *Proceedings of the IEEE Symposium on FPGAs for Custom Computing Machines*.

Hwu, W. W., Haab, G. E., Holm, J. G., Lavery, D. M., Mahlke, S. A., Chen, W. Y., Chang, P. P., Warter, N. J., Bringmann, R. A., Ouellette, R. G., Hank, R. E., and Kiyohara, T. 1993. The superblock: An effective structure for vliw and superscalar compilation. *J. Supercomput. 7*, 1–2, 229–248.

Kastner, R., Memik, S. O., Bozorgzadeh, E., and Sarrafzadeh, M. 2001. Instruction generation for hybrid reconfigurable systems. In *Proceedings of the International Conference on Computer-Aided Design*.

Kramer, H. and Rosenstiel, W. 1990. System synthesis using behavioral descriptions. In *Proceedings of the European Design Automation Conference*.

Lee, D. T. 1996. “Computational geometry”. In *The Computer Science and Engineering Handbook*, Chap. 6, Allen B. Tucker, Ed. CRC Press, Boca Raton, FL.

Lee, C., Potkonjak, M., and Mangione-Smith, W. H. 1997. Mediabench: A tool for evaluating and synthesizing multimedia and communications systems. In *Proceedings of the International Symposium on Microarchitecture*.

Lucent Technologies. Lucent technologies announces high-speed communications cores for customizing ORCA FPGAs.

Mahlke, S. A., Lin, D. C., Chen, W. Y., Hank, R. E., and Bringmann, R. A. 1992. Effective compiler support for predicated execution using the hyperblock. In *Proceedings of the International Symposium on Microarchitecture*.

McFarland, M. C. 1986. Using bottom-up design techniques in the synthesis of digital hardware from abstract behavioral descriptions. In *Proceedings of the Design Automation Conference*.

McFarland, M. C., Parker, A. C., and Camposano, R. 1988. Tutorial on high-level synthesis. In *Proceedings of the Design Automation Conference*.

Monteiro, J., Devadas, S., Ashar, P., and Mauskar, A. 1996. Scheduling techniques to enable power management. In *Proceedings of the Design Automation Conference*.

Musoll, E. and Cortadella, J. 1995. Scheduling and resource binding for low power. In *Proceedings of the International Symposium on System Synthesis*.

Pangrle, B. M. and Gajski, D. 1987. Design tools for intelligent silicon compilation. *IEEE Trans. Comput.-Aided Des. 6*, 6, 1098–1112.

Parker, A. C., Pizarro, J., and Mlinar, M. 1986. Maha: A program for datapath synthesis. In *Proceedings of the Design Automation Conference*.

Paulin, P. G. and Knight, J. P. 1987. Force directed scheduling in automatic data path synthesis. In *Proceedings of the International Conference on Computer Design*.

Paulin, P. J. and Knight, J. P. 1989. Force directed scheduling for behavioral synthesis of asics. *IEEE Transactions on Comput.-Aided Des. 8*, 6, 661–679.

Potasman, J., Lis, J., Nicolau, A., and Gajski, D. 1990. Percolation based synthesis. In *Proceedings of the Design Automation Conference*.

Schreiber, R., Aditya, S. G., Rau, B. R., Mahlke, S., Kathail, V., Cronquist, D., and Sivaraman, M. 2002. Pico-npa high-level syntesis of nonprogrammable hardware accelerators. *J. VLSI Signal Process. 31*, 2, 127–142.

Shiue, W. and Chakrabarti, C. 2000. Low-power scheduling with resources operating at multiple voltages. *IEEE Trans. Circ. Syst. II: Analog Digital Sign. Process. 47*, 6, 536–543.

So, B., Hall, M., and Diniz, P. 2002. A compiler approach to fast design space exploration in fpga-based systems. In *Proceedings of the Conference on Programming Language Design and Implementation.*

Srivastava, A., Memik, S. O., Choi, B. K., and Sarrafzadeh, M. 2003. Achieving design closure through delay relaxation parameter. In *Proceedings of the International Symposium on Computer Aided Design.*

Thomas, D. E., Lagnese, E. D., Walker, R. A., Nestor, J. A., Rajan, J. V., and Blackburn, R. L. 1990. *Algorithmic and Register Transfer Level Synthesis: The System Architect's Workbench*. Kluwer Academic Publishers, Norwell, MA.

Timmer, A. H. and Jess, J. A. G. 1995. Exact scheduling strategies based on bipartite graph matching. In *Proceedings of the European Design and Test Conference.*

Wazlowski, M., Agarwal, L., Lee, T., Smith, A., Lam, E., Athanas, P., Silverman, H., and Gosh, S. 1993. Prism-ii compiler and architecture. In *Proceedings of the IEEE Workshop on FPGAs for Custom Computing Machines.*

Wong, J. L., Megerian, S., and Potkonjak, M. 2002. Forward-looking objective functions: Concepts and applications in high level synthesis. In *Proceedings of the Design Automation Conference.*

Xilinx, Inc. Visit the Web site www.xilinx.com/apps/appsweb.htm.

Received January 2003; revised August 2003, October 2003; accepted November 2003

Combinatorial Techniques for Mixed-Size Placement

S. N. ADYA and I. L. MARKOV
University of Michigan, Ann Arbor

While recent literature on circuit layout addresses large-scale standard-cell placement, the authors typically assume that all macros are fixed. Floorplanning techniques are very good at handling macros, but do not scale to hundreds of thousands of placeable objects. Therefore we combine floorplanning techniques with placement techniques to solve the more general placement problem. Our work shows how to place macros consistently with large numbers of small standard cells. Proposed techniques can also be used to guide circuit designers who prefer to place macros by hand.

We address the computational difficulty of layout problems involving large macros and numerous small logic cells at the same time. Proposed algorithms are evaluated in the context of wirelength minimization because a computational method that is not scalable in optimizing wirelength is unlikely to be successful for more complex objectives (congestion, delay, power, etc.)

We propose several different design flows to place mixed-size placement instances. The first flow relies on an arbitrary black-box standard-cell placer to obtain an initial placement and then removes possible overlaps using a fixed-outline floorplanner. This results in valid placements for macros, which are considered fixed. Remaining standard cells are then placed by another call to the standard-cell placer. In the second flow a standard-cell placer generates an initial placement and a force-directed placer is used in the engineering change order (ECO) mode to generate an overlap-free placement. Empirical evaluation on *ibm* benchmarks shows that in most cases our proposed flows compare favorably with previously published mixed-size placers, Kraftwerk, and the mixed-size floor-placer proposed at the 2003 Conference on Design, Automation, and Test in Europe (DATE 2003), and are competitive with mPG-MS.

Categories and Subject Descriptors: B.7.2 [**Integrated Circuits**]: Design Aids

General Terms: Algorithms, Design, Performance

Additional Key Words and Phrases: VLSI, placement, floorplanning

1. INTRODUCTION

During the last few decades, academia and industry have invested considerable effort in research on physical design for very large-scale integration (VLSI) [Sherwani 1999]. Through the integration of multiple optimization techniques, design methods and high-performance computer-aided design (CAD) software

Authors' addresses: EECS Department, University of Michigan, 1301 Beal Ave., Ann Arbor, MI 48109-2122; email: {sadya,imarkov}@eecs.umich.edu.

for integrated circuits (ICs) were developed. However, the growing size of ICs lead to frequent changes to common design flows. Recently, design reuse was introduced as a way to (i) tame the complexity of circuit design for deep submicron technologies, and (ii) improve time-to-market. This trend is further accelerated with the use of hardware description languages and high-level synthesis. Indeed, several current industrial initiatives provide infrastructure and training for the reuse of intellectual property (IP), and also facilitate business models based on IP reuse.

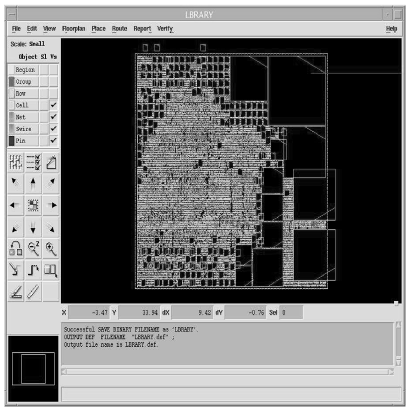
Reuse of design IP is important for multi-million-gate application-specific integrated circuits (ASICs) and considered an integral part of the system-on-chip (SoC) design style, that is critical for graphics cards, communication chips, etc. Design IP blocks may implement algorithms or signal transforms, and may contain "canned" table lookups or embedded RAM.

During physical design, IP design blocks appear as black-box *macros*, that is, blocks of logic with known function and known geometric and electrical properties, but no structural description of their inner workings. Such macros may or may not have flexible geometries, but in any case are considered parts of design. In classical physical design flows a circuit is first partitioned, then floorplanned, and finally standard-cell placement is performed in each partition. This was necessary primarily because older placers, for example, those based on simulated annealing, did not scale very well. However the scalability of min-cut placers dramatically improved after the multilevel partitioning breakthrough in 1997 [Alpert et al. 1997; Karypis et al. 1997; Caldwell et al. 2000a]. In addition to having near-linear runtime, placers based on recursive bisection *perform circuit partitioning* and, if the cut-lines are allowed to move, also *perform floorplanning*. Yet macro-placement is not supported in these placers, mainly because large macros that contain more than several percent of layout area introduce considerable discreteness in the solution space and may be difficult to handle within standard recursive min-cut bisection.

Reusing black-box macros in physical design still remains a challenge and existing commercial tools often require help from human designers. For example, the Cadence QPlace manual [Cadence 2000] mentions that the addition of macros may slow down otherwise fairly efficient placement of standard cells and the results may be inferior to what human designers can achieve. Cadence Silicon Ensemble (SEDSM) recommends the following flow for circuits with macros:

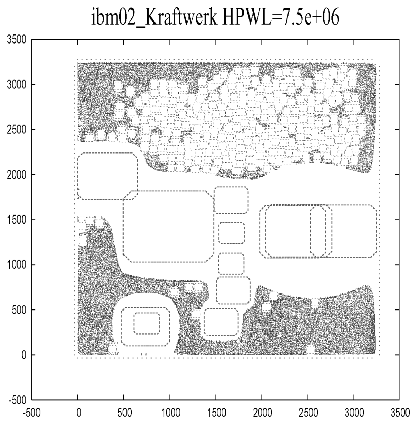
—Locations of macros are found by invoking block placement. Macros may have overlaps and may not fit in layout area.
—Human designer manually removes any overlaps between macros.
—Macros are now considered fixed.
—QPlace is called to place standard cells.

Figure 1(a) shows the placement of the ibm02 design (see Section 5), produced with the Cadence SEDSM flow recommended for circuits with a large number of macros. As seen in the figure, there is a large amount of overlaps between macros and the designer is expected to remove these overlaps manually. If the design is given directly to the SEDSM placer, QPlace, a legal

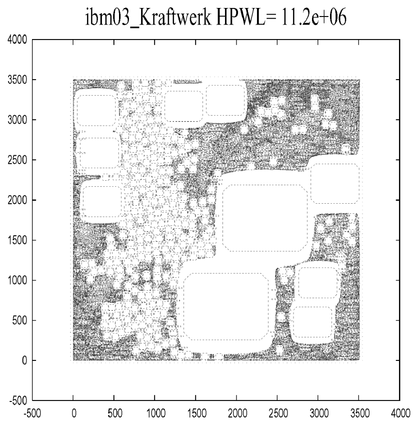


```
X -3.47 Y 33.94 dX 9.42 dY -0.76 Sel 0
Successful SAVE BINARY FILENAME as 'LBRARY'.
OUTPUT DEF  FILENAME  "LBRARY.def" ;
Output file name is LBRARY.def.
```

(a)



(b)



(c)

Fig. 1. Figure 1(a) shows a design with 19601 cells (ibm02) design placed by Cadence SEDSM recommended flow for designs with a large number of macros. There are overlaps between macros which the designer is expected to remove manually. Figure 1(b) shows the Kraftwerk placement for the same design. Again, there are significant overlaps, which have to be removed. Figure 1(c) shows the Kraftwerk placement for a design with 23136 cells (ibm03). The overlaps between macros is much smaller than those shown in 1(b) and can probably be removed by simple techniques.

placement is produced, but the runtimes and solution quality suffer. However, a new version of SEDSM is currently in beta-testing and implements a different macro-placement flow, achieving better results [Varadrajan and DeLendonck 2002]. Recently acquired by Cadence, Silicon Perspective developed the First Encounter tool which performs system-on-chip physical-prototyping and hierarchical physical design. First Encounter allows full-chip physical prototyping and emphasizes early floorplanning. Figure 1(b) shows the placement of the same ibm02 design produced by the force-directed placer Kraftwerk [Eisenmann and Johannes 1998]. This placement also has a large amount of overlap between macros. Figure 1(c) shows a Kraftwerk placement for design ibm03. As seen in the figure, there are no significant overlaps between macros in this placement and relatively simple techniques should be able to legalize such a placement. However, Kraftwerk does not always produce such nonoverlapping placements and more sophisticated legalization techniques are required.

In addition to physical design with IP blocks, mixed-size placement techniques are relevant in the context of physical synthesis, where layout starts before the netlist is fully synthesized. While our work does not address synthesis, the proposed techniques may be useful in physical synthesis tools that operate at the chip level.

Previous published work on mixed-size placement can be broadly classified into two approaches: continuous optimization techniques and combinatorial optimization techniques. Continuous optimization techniques like force-directed approaches work well with less constrained designs having relatively large amount of white-space [Eisenmann and Johannes 1998; Mo et al. 2000]. On the other hand, combinatorial techniques are particularly promising on constrained designs with less white-space [Nag and Chaudhary 1999]. Published works [Nag and Chaudhary 1999; Vijayan 1991] have focused on overlap removal for macros only and did not consider mixed-size placement. An entirely different approach was pursued in Chang et al. [2003]. Their placer mPG-MS was based on an earlier tool mPG, which recursively clusters the netlist to build a hierarchy. The top-level netlist of approximately 500 clusters is placed using simulated annealing (SA), and then the placement is gradually refined by unclustering the netlist and improving the placement of smaller clusters by SA. mPG-MS contributes a structure of bins, in which large and small blocks are placed during course placement. The coarse placement is necessarily overlap-free for big objects, but small objects must be further re-placed by a detail placer. A significant effort is expended to check for overlap during refinement and legalize possible violations. A recent work [Choi and Bazargan 2003] also dealt with the mixed-size placement problem. The authors proposed a mixed-size placement flow which combines a hierarchical simulated annealing-based floorplanner with partitioning-based placement techniques to handle the mixed-size placement problem. Their method starts with a netlist and a fixed-floorplan area. At each level of partitioning, "large" hard macros are extracted from the netlist and the rest of the standard-cells and small macros are partitioned into a number of "soft" modules using a min-cut partitioner. The mixed hard/soft modules are floorplanned using a slicing floorplanner. This is performed recursively until the modules contain fewer than 30 gates. The method employs area migration techniques to satisfy the fixed-outline constraints. However, this method does not produce completely legal placements, with large overlaps remaining between macros. The reader is referred to the book by Sarrafzadeh et al. [2002] for a detailed background discussion of mixed-size placement.

The main contribution of our work is a methodology to place designs with numerous macros by combining floorplanning and standard-cell techniques. The proposed design flow is as follows:

—An arbitrary black-box (no access to source code required) standard-cell placer generates an initial placement.

—To remove overlaps between macros, a physical clustering algorithm constructs a fixed-outline floorplanning instance.
—A fixed-outline floorplanner [Adya and Markov 2001] generates valid locations of macros.
—With macros considered fixed, the black-box standard-cell placer is called again to place small cells.

This design flow provides a somewhat new "killer-application" for the many floorplanning techniques developed in the last 5 years (e.g., Lin and Chang [2001]). Indeed, we do not insist on using a particular floorplan representation, but rather emphasize floorplanning as a step in large-scale placement with macros.

We also propose a second design flow combining a black box standard-cell placer and a force-directed placer. The proposed design flow is as follows:

—An arbitrary black-box (no access to source code required) standard-cell placer generates an initial placement.
—A force-directed placer [Eisenmann and Johannes 1998] is used in engineering change order (ECO) mode to remove the overlaps while changing the initial placement minimally.

Depending upon the design requirements and characteristics of the design, either of our flows can be used to produce high-quality placements of mixed-size designs.

We notice that existing academic placers Capo [Caldwell et al. 2000a], Dragon 2000 [Wang and Sarrafzadeh 2000], Feng Shui [Yildiz and Madden 2001], and Spade [Dutt 2000] cannot process movable macros. In fact, *all macros are removed* in the placement benchmarks described in Wang and Sarrafzadeh [2000] (produced from the ISPD 98 circuit benchmarks), and all cells are artificially made one by one. Therefore, we derived new placement benchmarks from the original ISPD 98 circuits, preserving macros and the areas of all cells. Having converted the benchmarks into Cadence LEF/DEF format, we compared the performance of our methods to the Cadence commercial placer, QPlace, Kraftwerk, mPG-MS, and the mixed-size placement flow proposed in Choi and Bazargan [2003].

The remaining part of the article is organized as follows. Section 2 covers previous work relevant to force-directed placement and fixed-outline floorplanning. Two new design flows for macro placement are proposed in Sections 3 and 4. Section 5 presents empirical validation of our work, and future directions are discussed in Section 6. Section 7 concludes our work.

2. PREVIOUS WORK

In this section we outline the relevant background for our study. We briefly describe the top-down recursive bisection-based placement framework, a generic force-directed placement and floorplanning algorithm, and a fixed-outline floorplanning algorithm. All these algorithms are used in our proposed mixed-size placement flows.

2.1 Top-Down, Recursive Bisection-Based Placement

Top-down placement algorithms seek to decompose a given placement instance into smaller instances by subdividing the placement region, assigning modules to subregions, reformulating constraints, and cutting the netlist hypergraph [Caldwell et al. 2000a]. Such a netlist decomposition is typically done with the min-cut objective. Each hypergraph partitioning instance is induced from a rectangular region, or block in the layout. A block corresponds to (i) placement region with allowed locations, (ii) a collection of cells to be placed in this region, (iii) all nets incident to the modules in the region, and (iv) fixed-terminals which are cells outside the region. The top-down placement process can be viewed as a sequence of passes where each pass examines all blocks and, if required, divides them into two smaller blocks using min-cut partitioning. In our work we used the top-down recursive bisection-based placer Capo [Caldwell et al. 2000a]. Capo uniformly distributes the available whitespace [Caldwell et al. 2003] around the core. However, if nonuniform distribution is required, fake unconnected filler cells [Adya et al. 2003a] can be used.

2.2 Kraftwerk: Generic Global Placement and Floorplanning

A force-directed method for global placement was introduced in Eisenmann and Johannes [1998]. Their global placer is called *Kraftwerk*. In addition to the well-known wirelength-dependent forces, Kraftwerk uses additional forces to reduce cell-overlaps and to consider the placement area. The wirelength-dependent quadratic objective function to minimize is described as follows. Let n be the number of movable cells in the circuit and (x_i, y_i), the coordinates of cell i. A placement of the circuit can be described by the $2n$-dimensional vector $\vec{p} = (x_1, .., x_i, \ldots, x_n, y_1, .., y_i, .., y_n)^T$. The circuit connectivity is modeled as a graph. Cells are modeled as vertices and nets are modeled as edges. Hyperedges are modeled as cliques. The cost of an edge is modeled as the squared Euclidean distance between its adjacent vertices multiplied with the weights of the edges. The squared Euclidean distance between cells i and j is $(x_i - x_j)^2 + (y_i - y_j)^2$. The objective function sums up the cost of all edges and can be written in matrix notation as

$$\frac{1}{2}\vec{p}^T C\vec{p} + \vec{d}^T\vec{p} + const.$$

This objective function is minimized by solving the linear equation system

$$C\vec{p} + \vec{d} = 0.$$

Additional constant forces were introduced in Eisenmann and Johannes [1998] to distribute the cells more evenly in the layout region:

$$C\vec{p} + \vec{d} + \vec{e} = 0.$$

The force vector $\vec{e}$ contains additional forces working on each cell in the x and y directions. These additional forces try to move the cells from high-density regions to low-density regions in the layout, thus attempting to reduce the overlaps. The algorithm described in Eisenmann and Johannes [1998] is iterative, which determines the additional forces according to the current placement.

In each iteration, the forces acting on the cells are assumed constant and are used to calculate a new placement. The new placement is the basis for the next iteration step and so on. Each step of the algorithm is called a *placement transformation*. The transformation step can be applied to fully overlapping placements as well as nearly legal placements. Thus, the algorithm renders itself very elegantly to ECO-style placement requirements.

It was argued in Eisenmann and Johannes [1998] that their algorithm is able to handle large mixed-size placement problems without treating macros and standard cells differently. However, from our experiments, we conclude that if applied from scratch on constrained mixed-size designs with less white-space, this algorithm frequently produces placements with large overlaps.

2.3 Fixed-Outline Floorplanning

A fixed-outline floorplanner was proposed in Adya and Markov [2001, 2003]. We describe the work briefly here.

A typical floorplanning formulation includes a set of blocks, which may represent circuit partitions in applications. Each block is characterized by *area* (typically fixed) and *shape-type*, for example, fixed rectangle, rectangle with varying aspect ratio, etc. Multiple aspect ratios can be implied by an IP block available in several shapes as well as by a hierarchical partitioning-driven design flow for ASICs [Sherwani 1999; Kahng 2000] where *only the number of standard cells* in a block (and thus the total area) is known in advance. A solution to such a problem, that is, a *floorplan*, specifies a selection of block shapes and overlap-free placements of blocks. Classical floorplanning minimizes a linear combination of area and wirelength. Among measures of circuit wirelength, the popularity of the half-perimeter wirelength (HPWL) function is due to its simplicity and relative accuracy before routing is performed. The HPWL objective gained relevance with the advent of multilayer over-the-cell routing, where more nets are routed with shortest paths [Kahng 2000]. In floorplanners based on simulated annealing (e.g., with the sequence-pair representation [Murata et al. 1996]), the typical choice of moves is straightforward.

As pointed out in Kahng [2000], and Caldwell et al. [2000a], modern hierarchical ASIC design flows based on multilayer over-the-cell routing naturally imply *fixed-die* placement and floorplanning, rather than the older *variable-die* style [Sherwani 1999], associated with channel routing, two layers of metal, and feedthroughs. In such a flow, each top-down step may start with a floorplan of prescribed aspect ratio and with blocks of bounded (but not fixed) aspect ratios. The modern floorplanning formulation proposed in Kahng [2000] is an inside-out version of the classical outline-free floorplanning formulation—the aspect ratio of the floorplan is fixed, but the aspect ratios of the blocks may vary.

2.3.1 *Sequence-Pair Floorplan Representation.*

An overwhelming majority of floorplanners rely on the simulated annealing framework [Sherwani 1999] but differ by internal floorplan representations.

The sequence-pair representation for classical floorplans consists of two permutations (orderings) of the N blocks [Murata et al. 1996]. The two permutations capture geometric relations between each pair of blocks. Recall that since

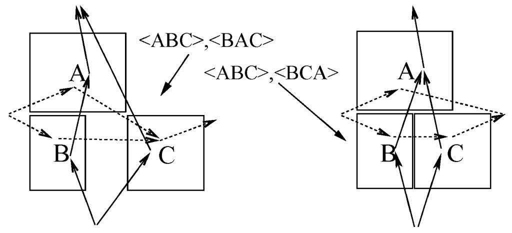


Fig. 2. Two sequence-pairs with edges of the horizontal (dashed) and vertical (solid) constraint graphs.

blocks cannot overlap, one of them must be to the left of or below the other, or both. In sequence-pair

$$(<\ldots,a,\ldots,b,\ldots>,<\ldots,a,\ldots,b,\ldots>) \Rightarrow a \text{ is to the left of } b, \qquad (1)$$

$$(<\ldots,a,\ldots,b,\ldots>,<\ldots,b,\ldots,a,\ldots>) \Rightarrow a \text{ is above } b. \qquad (2)$$

In other words, every two blocks constrain each other in either the vertical or horizontal direction. The sequence-pair representation is shift-invariant since it only encodes pairwise relative placements. Therefore, placements produced from sequence-pairs must be aligned to given horizontal and vertical axes, for example, $x = 0$ and $y = 0$.

The original work on sequence-pair [Murata et al. 1996] proposed an algorithm to compute placements from a sequence-pair by constructing horizontal (H) and vertical (V) constraint graphs. The H and V graphs have $N + 2$ vertices each—one for each of N block, plus two additional vertices: "the source" and "the sink." For every pair of blocks a and b there is a directed edge $a \rightarrow b$ in the H graph if a is to the left of b according to the sequence-pair (Formula 1). Similarly there is a directed edge $a \rightarrow b$ in the V graph if a is above b according to the sequence-pair (Formula 2)—and exactly one of the two cases must take place. Vertices that do not have outgoing edges are connected to the sink, and vertices that do not have incoming edges are connected to the source. Both graphs are considered vertex-weighted; the weights in the H graph represent horizontal sizes of blocks, and the weights in the V graph represent vertical sizes of blocks. Sources and sinks have zero weights.

Block locations are the locations of block's lower left corners. The x locations are computed from the H graph, and y locations are computed from the V graph *independently*. Therefore, we will only discuss the computation of the x locations. One starts by assigning location $x = 0$ to the source. Then, the H graph is traversed in a topological order. To find the location of a vertex, one iterates over all incoming edges and maximizes the sum of the source location and source width. Figure 2 illustrates the algorithm on two examples. The worst-case and average-case complexity of this algorithm is $\Theta(n^2)$, since the two graphs, together, have a fixed $\Theta(n^2)$ number of edges, and topological traversals take linear time in the number of edges.

Sequence-pairs can be used to floorplan hard rectangular blocks by simulated annealing [Murata et al. 1996; Murata and Kuh 1998; Tang et al. 2000; Tang and Wong 2001]. The moves are (i) random swaps of blocks in one of the two sequence-pairs, and (ii) rotations of single blocks. Sequence-pairs are modified in constant time, but need to be reevaluated after each move. No incremental evaluation algorithms have been reported; therefore, the annealer spends most of the time evaluating sequence-pairs.

The sequence-pair representation and necessary algorithms have been extended to handle fixed blocks [Murata and Kuh 1998] as well as arbitrary convex and concave rectilinear blocks [Fujuyoshi and Murata 1999]. Recently, the original $O(n^2)$-time evaluation algorithm Murata et al. [1996] has been simplified and sped up to $O(n \log(n))$ by Tang et al. [2000], and then to $O(n \log(\log(n)))$ [Tang and Wong 2001]. Importantly, those algorithms do not change the semantics of evaluation—they only improve runtime, and lead to better solution quality by enabling a larger number of iterations during the same period of time. While O-trees [Pang et al. 2000] and corner block lists [Hong et al. 2000] can be evaluated in linear time, the difference in complexity is dwarfed by implementation variations and tuning, for example, the annealing schedule. The implementation reported by Tang et al. [Tang and Wong 2001] seems to outperform most known implementations, suggesting that the sequence-pair is a competitive floorplan representation.

All three sequence-pair evaluation algorithms are based on the following theorem [Tang et al. 2000]: the x-span of the floorplan to which sequence pair (S_1, S_2) evaluates is equal to the length of the longest common weighted subsequence of S_1 and S_2, where weights are copied from block widths. An analogous statement about the y-span deals with the longest common subsequence of S_1^R and S_2 , where ^{R} denotes the "reversed" sequence and weights are copied from block heights. Moreover, the computations of x and y locations of all blocks can be integrated into the longest common subsequence computations.

2.3.2 *Floorplan Slacks.* The notion of slack can be used with any of above mentioned sequence pair evaluation algorithms and potentially other floorplan representations [Adya and Markov 2003]. Each block in a floorplan has two types of slacks: horizontal slack and vertical slack. Slack of a block in a floorplanning instance represents the distance (in a particular dimension) at which this block can be moved without changing the outline of the current floorplan. Blocks with zero slacks in a particular dimension must lie on critical paths in the relevant constraint graph.

We will base our discussion on the horizontal slack. The discussion on vertical slack is analogous. As shown in Figure 3, horizontal slacks can be computed with any floorplan representation that can be evaluated left-to-right and right-to-left. Once the x-size of the floorplan is computed by packing left-to-right, one can repack it right-to-left. The horizontal slack of a block is the difference between the block's locations produced by those two packings. The floorplanner Parquet [Adya and Markov 2001] uses the sequence-pair representation because of the simplicity of the representation.

Once slacks for each block are known, they can be used in move selection. The rationale here is to reduce the floorplan size in a given dimension (x or y) without impairing the hill-climbing abilities of simulated annealing. The new mechanism is combined with pairwise swaps and block rotations that are typically used in sequence-pair-based annealers. If a move (such as pairwise swap) does

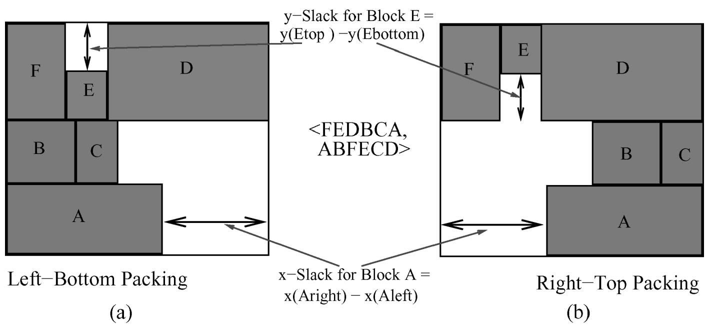


Fig. 3. Slack computation. In (a) the floorplan is evaluated left-to-right and bottom-to-top. In (b) the floorplan is evaluated right-to-left and top-to-bottom. The slack for each block is the difference between its positions in the two evaluations.

not involve at least one block with zero slack in a given dimension, then the floorplan size in that dimension cannot decrease after the move. This is because such a move cannot improve critical paths or, equivalently, longest common subsequences [Tang et al. 2000; Tang and Wong 2001]. Therefore *move selection is biased toward blocks having zero slack in at least one dimension*. Of those blocks, the ones with large slack in the other dimension are often good candidates for single-block moves, such as rotations and gradual (discrete or continuous) changes of aspect ratio. Blocks with zero slack in both the directions, especially small blocks, are good candidates for a new type of move, in which a block is moved simultaneously in both sequence pairs to become a neighbor of another block (in both sequences, and, thus in placement). One possible heuristic is to move a critical block C next to a block L with as large a slack as possible, since large slacks imply that white-space can be created around L.

2.3.3 *Handling Soft Blocks.* We can also use slack-based move types to change aspect ratios of soft blocks [Adya and Markov 2003]. During annealing, at regular intervals, a block with low (preferably zero) slack in one dimension and large slack in the other dimension is chosen. The height and the width of such a block are changed within allowable limits so that its size in the dimension of smaller slack is reduced (to increase the slack). Such moves are greedily applied to all soft blocks in the design.

2.3.4 *Wirelength Minimization.* In classical floorplanning, the global objective is to minimize wirelength and total area of the design. This implies multiobjective minimization. Typically, most simulated annealing-based floorplanners use a linear combination of area and wirelength as an objective for the annealer.

Additional moves can be designed to improve the wirelength [Adya and Markov 2003]. For a given block a, we calculate, using analytical techniques, its "ideal" location that would minimize quadratic wirelength of its incident wires N. We determine the ideal location (x_a, y_a) of block a which minimizes the following function:

$$\sum_{N \in a} \sum_{v \in N} (x_v - x_a)^2 + (y_v - y_a)^2.$$

The ideal location (x_a, y_a) of block a is simply the average of the position of all modules connected to block a. We then identify the block b closest to the ideal location. This is done by expanding a circle centered at the ideal location and identifying the closest block b. We then attempt to move block a in the sequence pair so that in both sequences it is located next to b. As explained earlier, we evaluate the four possible ways to do that, and choose the best. Thus an attempt is made to move a close to its ideal location to minimize quadratic wirelength.

2.3.5 *Fixed-Outline Floorplanning.* Fixed-outline floorplanning can be performed using simulated annealing, taking advantage of new types of moves that are based on the notion of *floorplan slack* [Adya and Markov 2001]. The following notation will be used in the floorplanning formulations. For a given collection of blocks with total area A and given *maximum percent of white-space* γ, we construct a fixed outline with aspect ratio $\alpha \geq 1$[1]:

$$H_* = \sqrt{(1+\gamma)A\alpha} \quad W_* = \sqrt{(1+\gamma)A/\alpha}.$$

Aside from driving the annealer by area minimization, we can consider the following objective functions: (i) the sum of the excessive length and width of the floorplan, (ii) the greater of the two. Denoting the current height and width of the floorplan by H and W, respectively, we define these functions as

$$\textbf{(i)}\ \max\{H - H_*, 0\} + \max\{W - W_*, 0\}, \quad \textbf{(ii)}\ \max\{H - H_*, W - W_*\}.$$

The choice of these functions is explained by the fact that the fixed-outline constraint is satisfied when each of those functions takes value zero or less. For this reason we cannot consider the product of fixed outline violations.

Figure 4 shows the evolution of the fixed-outline floorplan during simulated annealing with slack-based moves. The scheme works as follows. At regular time intervals (during area-minimizing simulated annealing) the current aspect ratio is compared to the aspect ratio of the desired outline. If the two are sufficiently different, then the slack-based moves described earlier are applied to bias the current aspect ratio in the needed direction. For example, if the width needs to be reduced, then choose the blocks in the floorplan with smallest slack in the x dimension and insert them above or below the blocks with largest slack in the y dimension. These moves have better chances of reducing the area and improving the aspect ratio of the current floorplan at the same time. Using such repeated corrections, the structure of the floorplan is biased toward the aspect ratio of the fixed outline.

While a number of works on floorplanning have discussed floorplan constraints, the results in Adya and Markov [2003] and Adya and Markov [2001] empirically demonstrated high ratios of successes to failures in the flow from Figure 4.

3. MIXED-SIZE PLACEMENT FLOW 1

Our first proposed flow for mixed-size placement requires a black-box standard-cell placer that can place cells of equal height in rows that consist of cell sites,

[1]The restriction of $\alpha \geq 1$ is imposed without loss of generality since our floorplanner can change orientations of individual blocks.

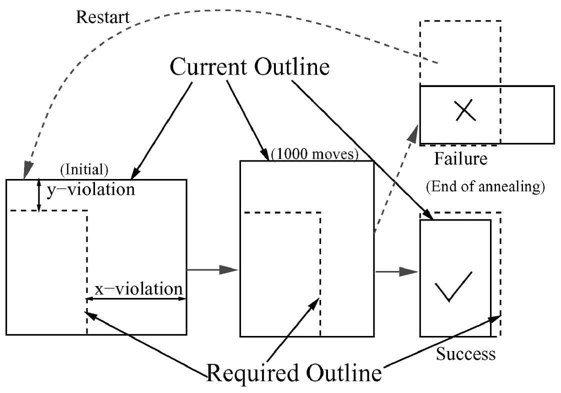


Fig. 4. Snapshots from fixed-outline floorplanning. The number of annealing moves is fixed, but if the evolving floorplan fits within the required fixed-outline, annealing is stopped earlier. If at the end of annealing the fixed-outline constraints are not satisfied, it is considered a failure and a fresh attempt is made.

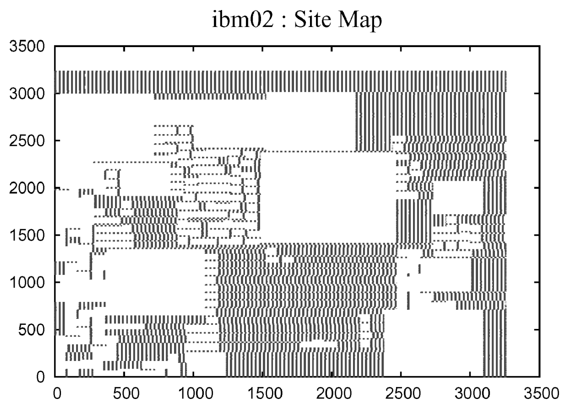
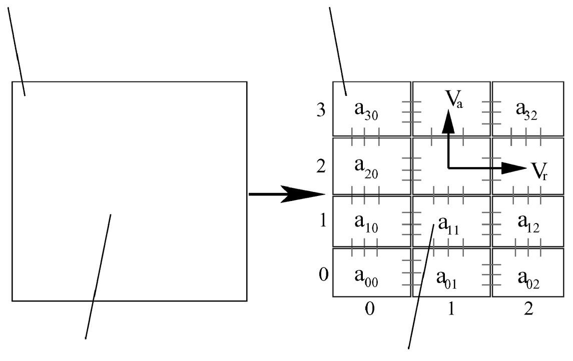


Fig. 5. Map of cell sites for the ibm02 design with all the macros marked as fixed. Sites under the macros are removed.

along the lines of the data-model implied by Cadence LEF/DEF. We also require that the placer be able to handle fixed cells/pins and be able to handle rows consisting of contiguous subrows. By removing cell sites from a subrow and splitting the subrow into two subrows, one can model the effect of fixed macros (because pins of fixed macros are fixed as well). For example, the site map in Figure 5 corresponds to the placement in Figure 10(c). Our flow also uses a fixed-outline floorplanner described in Section 2.3. While our floorplanner uses the sequence-pair representation, a variety of other floorplan representations can be used.

3.1 Shredding Macro Cells

A hierarchical recursive bisection-based placer has trouble handling mixed-size netlists [Sarrafzadeh et al. 2002] because of the large variations in the cell sizes. We get around this inherent problem by shredding all the macros to make



(a)

case:
$V_a \uparrow V_r \rightarrow$: orient = N
$V_a \downarrow V_r \leftarrow$: orient = S
$V_a \leftarrow V_r \uparrow$: orient = W
$V_a \rightarrow V_r \downarrow$: orient = E
$V_a \uparrow V_r \leftarrow$: orient = FN
$V_a \downarrow V_r \rightarrow$: orient = FS
$V_a \rightarrow V_r \uparrow$: orient = FW
$V_a \leftarrow V_r \downarrow$: orient = FE
end case;

(b)

Fig. 6. (a) A macro is shredded into cells of minimal height, connected by fake wires. To find the orientation of the macro from locations of subcells, the relative locations of subcells $a_{i,j}$, $a_{i+1,j}$, and $a_{i,j+1}$ are analyzed for every eligible (i, j). (b) The case analysis in terms of vectors V_a and V_r in final placement. "N," "S," "W," "E" stand for "North," "South," "West," and "East," respectively. "F" stands for "Flipped." Any net connected to a macro pin is propagated to the respective subcell as shown.

the netlist more homogeneous in terms of cell sizes. The DOMINO detailed placer introduced the idea of shredding large cells to simplify placement [Doll et al. 1994]. To apply this technique in global placement, one must additionally handle cell orientations and remove cell overlaps (other than by left-to-right packing).

Our flow starts with a preprocessing step during which all macros are shredded into a number of smaller cells of minimal height. The number of these cells is determined by the area of the macro and the width of subcells. A macro shredded into subcells is shown in Figure 6. A subcell with row index i and column index j may be identified as $a_{i,j}$, and its immediate neighbors are $a_{i-1,j}$, $a_{i+1,j}$, $a_{i,j-1}$, and $a_{i,j+1}$. Fake two-pin nets are added between neighboring subcells to ensure that subcells are placed close to each other when wirelength is minimized. The number of fake nets added between each pair of subcells determine how strongly the subcells are tied to each other. We add three fake nets between each connected, neighboring subcells. The total number of faked wires depends on the width of the subcells. A cleverly implemented placer could handle the faked wires implicitly, for example, using net weights. In any case, a large-scale global placer with near-linear runtime (e.g., a fast min-cut placer) should be able to handle the increased number of wires. The Capo placer [Caldwell et al. 2000a] we use is scalable enough. The shredding procedure can be viewed as the equivalent of descending one level of hierarchy in a hierarchical design by flattening the macro. If the subcells of the macro are not placed close to each other in the placement of the flattened design then it implies that the macro was not formed properly, that is, the clustering technique employed to form the macro did not work very well. Artificially shredding the macro makes the new placement problem more homogeneous and thus a finely tuned min-cut-based placer can handle the shredded design better than handling the original placement problem with macros.

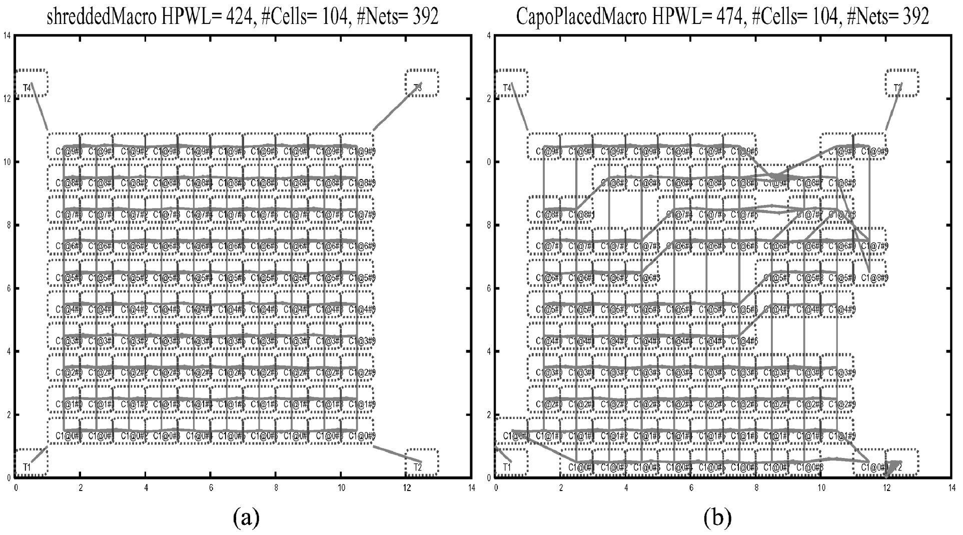

shreddedMacro HPWL= 424, #Cells= 104, #Nets= 392
CapoPlacedMacro HPWL= 474, #Cells= 104, #Nets= 392
T1
T2
T3
T4
(a)
(b)

Fig. 7. A design with only one macro and four terminals. (a) The macro shredded into subcells. The subcells are placed at ideal locations. The fake nets connecting them form a regular grid structure. (b) The shredded design placed by Capo followed by detailed placement. The subcells are placed close to each other and also maintain the initial grid structure (in (a)) on average.

The shredded design with the fake nets is placed using the global placer Capo. The resulting placement does not immediately imply the locations of the original macros, because the macros are shredded. The center-location of a given macro is determined by averaging the locations of all subcells of that macro. Since the subcells of a macro are connected in a regular grid structure, a good placer will ensure that the subcells are placed close to each other and in the original grid-like structure. Determining the orientation of the macros which is globally consistent with the placement is very important. A top-down global placement methodology that handles large macros by fixing macros in partitions as soon as the macros become too large for the partition has problems determining the orientation of individual macros. We developed a heuristic to determine the orientation of the macro using the initial placement information. The heuristic is based on the relative placement of each subcell with respect to its immediate neighbors. Namely, the placement of subcell $a_{i,j}$ is compared with the placements of $a_{i+1,j}$ and $a_{i,j+1}$. This is illustrated in Figure 6, where two vectors are computed for a given cell and then analyzed to produce one of eight possible orientation types. For each macro, a score table is maintained which records the number of subcells placed in a particular orientation. The orientation of the macro is chosen according to the highest score (if several orientations have comparably high scores, then we cannot conclude the orientation with certainty). The rationale is that the extra nets added while shredding will, in many cases, help the macro to approximately maintain its shape. Figure 7 shows an example design with only one macro and four terminals. Figure 7(a) shows the macro shredded into subcells and connected in a regular grid-like fashion by fake wires. The subcells are placed at ideal locations. Figure 7(b) is

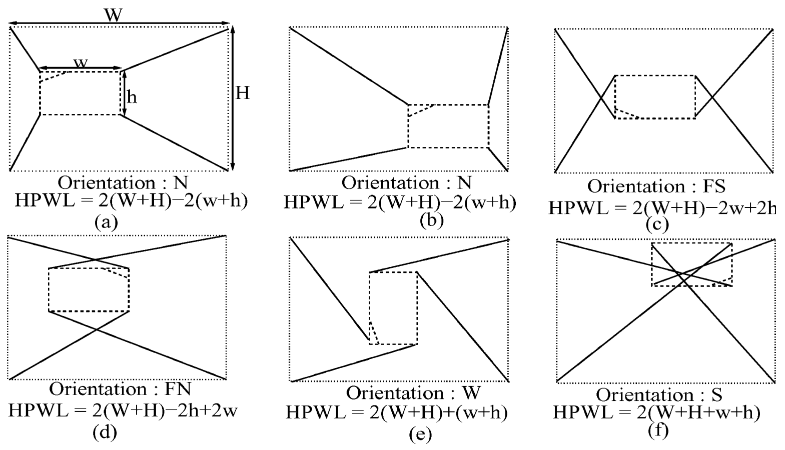


Fig. 8. Five out of eight orientations of a macro whose corners are tied to the corners of the layout region; the orientation is N in (a) and (b). The (linear) length of faked wires depends only on the orientation and not on the location of the macro, as long as the macro is placed entirely within the layout region. The desired orientation (N in this example) is found by wirelength minimization.

the shredded design placed by Capo followed by detailed placement. The subcells are placed close to each other and maintain the initial grid structure on average. From the placement of the shredded design, we deduce that the macro is placed in the north orientation.

Thus, a crude placement (with orientations) is obtained by placing the shredded design. Since the standard cells were placed by using wirelength-minimization, highly connected cells will be close to each other, but macros may overlap with each other and may not be placed entirely inside the layout region. Figure 10(a) shows the placement of the ibm02 circuit produced as explained above.

While our technique allows one to deduce the prevailing orientation of a macro or observe that there is no prevailing orientation, some macros may only be placeable in one orientation. Such a constraint can be ensured by tying the corners of the macro (i.e., the respective subcells after shredding) to the corners of the layout by strong (heavy) faked wires, as shown in Figures 8(a) and 8(b). During the minimization of HPWL, for example, by recursive min-cut bisection, the orientation of the macro will be preserved, and the quality of placement will not be affected. A formalization follows.

LEMMA. Placements that minimize HPWL in the original design subject to orientation constraints are in a one-to-one correspondence with unconstrained placements that minimize HPWL, including the fake wires that tie the corners of macros to the corners of the layout region (assuming sufficiently strong wires).

The lemma can be proven along the lines of Figure 8, where five out of eight possible orientations of a macro tied to the four corners of the layout region are shown. Note that this result does not apply to quadratic placement, and in that case all tied macros will be attracted to the center of the layout.

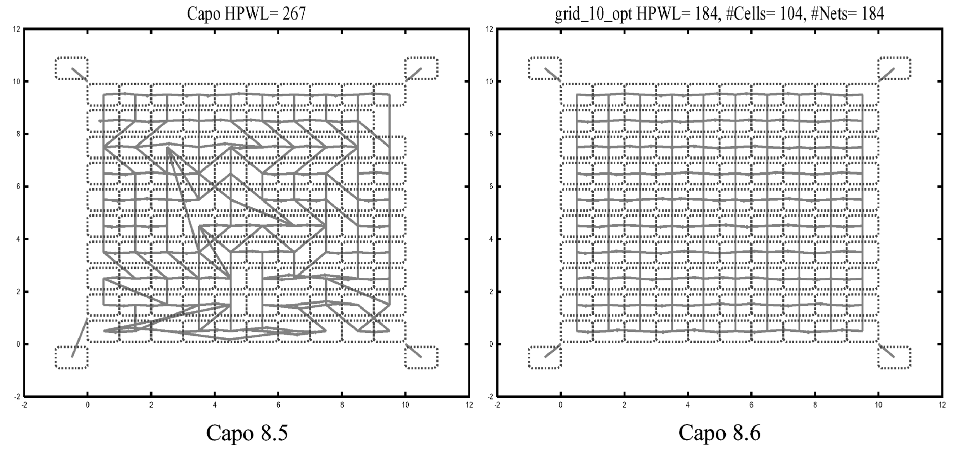


Fig. 9. Capo placements for designs with regular grid connectivity. Capo 8.5 produces suboptimal placements. Capo 8.6 produces the optimal placement for this design. There are four terminals connected to the four corner cells to anchor the design.

3.1.1 *Better Placement of Regular Netlists.* Observe that placement of the shredded netlists calls for placement of grid-graphs embedded into random-logic netlists. However, we discovered that the Capo 8.5 placer used in Adya and Markov [2002] performs poorly on grid-graphs, as shown in Figure 9 which illustrates an optimal and a suboptimal placement of a 10×10 grid with four fixed cells in the corners. This is hardly a surprise because generic standard-cell placers are known to perform badly on regular, data-path style designs [Dally and Chang 2000]. Our improvements to Capo allow it to better handle regular netlists without the loss of performance on random-logic netlists. These improvements are described below, and their implementation was contributed to Capo 8.6.

During each partitioning step with a vertical cut line, Capo 8.5 with default parameters uses a fairly large tolerance (of the order of 10–20%) in order to find better cuts. After a good cut is found, the geometric cut line is adjusted according to the sizes of partitions, with an equal distribution of white-space among the partitions. However, *if no white-space is available in the block*, this technique can cause cell overlaps. Namely, since cut-lines cannot cut through cell sites and since no "jagged" cut-lines are allowed, the set of partition balances that can be realized with a straight vertical cut-line and zero white-space is fairly discrete. Capo 8.5 simply rounds the current balance to the closest realizable and sets the geometric cut-line accordingly. When white-space is scarce, one of the resulting partitions may be overfull and the other may have artificially-created white-space. Only a relatively small number of cell overlaps can be created this way, but they can be spread through the whole core area. When used in the MetaPlacer shell, Capo 8.5 removes overlaps after global placement by a simple and very fast greedy heuristic. However, this heuristic increases wirelength.

In an attempt to reduce the number of overlaps, we revised the partitioning process in Capo. When a placement block is partitioned with a vertical cut-line, at first the tolerance is fairly large. As described previously, this allows Capo to

Table I. Wirelength Achieved by Several Placers on Regular Grids of Varying Size and with Varying White-Space (While Kraftwerk produces small wirelength on $n \times n$ grids, it often fails to converge to a solution on random-logic netlists with embedded grids.)

Circuit	#Nodes	#Nets	WS %	Optimal HPWL	Dragon HPWL	Kraftwerk HPWL	Capo default HPWL	Capo + repart HPWL
10×10	100	184	0	184	293	202	267	184
95×95	9025	17864	5	17884	39687	18302	21828	22764
100×100	10000	19804	0	19804	46066	20519	38352	21314
190×190	36100	71824	5	71864	175623	75384	90665	89814
200×200	40000	79604	0	79604	198182	82335	193167	100041

determine the location of the geometric cut-line by rounding to the nearest site. Furthermore, if the block has very little white-space, we then repartition it with a small tolerance in an attempt to rebalance the current partitions according to the newly defined geometric cut-line.

Another modification we implemented is related to terminal propagation. Normally, if a projection of a terminal's location is too close to the expected cut-line, the terminal is ignored by Capo in an attempt to avoid excessively speculative decisions. The proximity threshold is defined in percent of the current block size, and this parameter is called "partition fuzziness." For example, suppose that the y location of a terminal is within 9% of the tentative location of the horizontal cut-line. Then, with partition fuzziness of 10%, this terminal will be ignored during partitioning. Our studies of Capo performance on grids suggest that partition fuzziness should be tuned up, particularly for small blocks. For example, if a placement block has only three cell rows, then possible tentative locations of horizontal cut-lines are relatively far from the center. In a neighboring block that has not been partitioned yet, all cells are "located" at the center of the block, causing all connected terminals to propagate into one partition in the current block. To avoid this, we increased partition fuzziness to 33%.

The two changes described above improve the performance of Capo on the grid designs with 0% white-space by a factor of 2. The results for the performance of various placers on grid graphs [Adya et al. 2003b, 2004b] are reported in Table I.

3.2 Physical Clustering

The crude placement obtained from the above step may have overlapping macros as well as macros placed outside the layout region (Figure 10(a)). Such violations must be corrected without affecting the placement quality considerably. This can, in principle, be done by fixed-outline floorplanning, but the number of movable objects is unrealistically large (every standard cell is movable). We therefore construct a fixed-outline floorplanning instance through physical clustering based on locations of standard cells. Cells that are placed together are merged into soft clusters (i.e., the aspect ratio may vary). This is done by gridding the layout region and putting all the standard cells that physically fall within a grid region into a cluster. We recommend computing the dimensions of the grid based on the number of standard cells and macros in the design. However, in our experiments we used a grid of size 6 × 6 in order to speed up

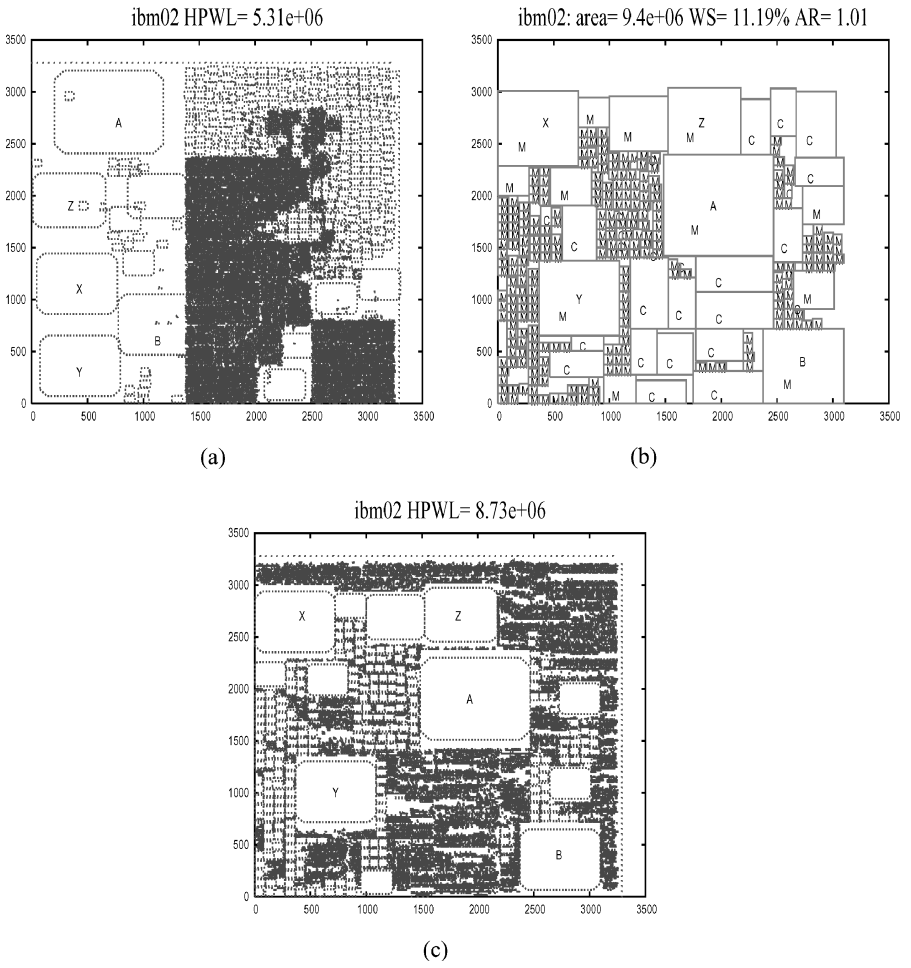


Fig. 10. Mixed-size placement Flow 1 explained in Section 3. (a) The placement (illegal) obtained after running Capo on ibm02 design with macros shredded into small cells. The locations of macros are determined by averaging the locations of subcells. Note that macro Z is not placed entirely within the layout region. Also, macro B overlaps with macro Y and standard cells. (b) A possible final fixed-outline floorplan of the same design. The floorplanning is done from scratch and no attempt is made to preserve the original locations of macros. Macros are marked with M and clusters of standard cells with C. Aspect ratios of macros are fixed and those of cell clusters vary between $^1/_2$ and 2. Observe that the vertical coordinates of three (A, X, and Y) out of five large macros are similar to those in part (a). (c) The final placement of ibm02. The locations of all macros are taken from the floorplan in part (b).

floorplanning. This grid worked well for smaller benchmarks, but appeared too coarse for larger benchmarks. The original macros are not clustered to anything, and their aspect ratio is allowed to change just as in the original placement formulation. For each cluster, the nets connecting only blocks within the cluster are discarded.

Since the design has been initially placed with small wirelength, the generated clusters contain strongly connected cells. Alternatively, one could use

Table II. Two Different Clustering Schemes to Form Floorplanning Instances (Connectivity-based greedy scheme groups highly connected objects together. Physical clustering groups objects, placed close to each other. In the clustered design, nets which connect only objects within a certain group are collapsed. The metric used to compare is the number of nets in the final clustered design. Physical clustering is more successful in reducing the number of nets crossing the groups compared to connectivity-based clustering.)

	Original netlist		Connectivity-based clustering			Physical clustering		
Circuit	#Nodes	#Nets	#Nodes	#Nets	Time	#Nodes	#Nets	Time
ibm01	12752	14111	196	**8732**	15.79s	282	**4689**	0.24s
ibm02	19601	19584	168	**14017**	42.26s	297	**6478**	0.39s
ibm03	23136	27401	272	**17069**	63.83s	332	**8723**	0.54s
ibm04	27507	31970	216	**20886**	84.05s	327	**9519**	0.58s
ibm05	29347	28446	28	**17280**	89.23s	37	**11630**	0.57s

connectivity-based clustering algorithms [Alpert et al. 1997; Karypis et al. 1997]. We compared the physical clustering scheme with one such simple greedy clustering scheme that consists of a series of passes. Each pass identifies pairs of connected vertices (the more connections between a pair of vertices, the more likely it is to be selected). Each pair is substituted with a cluster. The clustered vertices are removed, and all incident nets are connected to the new cluster. All nets whose pins are inside a single cluster are removed. Every such pass results in the reduction of the nodes in the netlist approximately by a factor of 2. The next pass is applied to the clustered netlist, and passes are continued until the desired reduction in size is achieved.

We compare this greedy connectivity-based clustering with physical clustering by the number of nets assuming approximately equal number of clusters (which is somewhat more rigorous than comparing the nets-to-clusters ratio). The results in Table II suggest that the physical clustering scheme is more successful in reducing the number of nets because even for larger numbers of clustered nodes it has lower numbers of nets. Of course, one could use more involved clustering schemes based on connectivity such as those using multiway min-cut partitioning. On the other hand, our physical clustering implicitly includes those algorithms. Another advantage is that our physical clustering based on the initial placement accounts for both netlist connectivity and the shapes of macros. Moreover, the initial placement can give the exact pin locations of the pins in the clustered cell. The initial placement is additionally used to construct an initial floorplan for simulated annealing. The blocks in this floorplan do not overlap, but may not fit into the desired outline. The initial placement run thus gives us information about macro locations, desired macro orientations, and highly connected cells to be clustered.

3.3 Fixed-Outline Floorplanning with Macros

The placement of macros obtained by placing the shredded netlist may have some overlaps and is in general not legal. We need to remove the overlaps between macros and ensure that they are all placed within the layout boundary. There are several possible options like using the approach in Nag and

Chaudhary [1999] for postplacement residual-overlap removal or force-directed approaches [Eisenmann and Johannes 1998]. However such approaches work well in less constrained designs with relatively large white-space.

As explained in Section 2.3.5, we use the fixed-outline floorplanner Parquet [Adya and Markov 2001] to satisfy the fixed-outline constraints imposed by fixed-die paradigm. This new version of Parquet is used to floorplan hard macros together with soft clusters of standard cells. The outline of the required floorplan is derived from the layout region and is used as a constraint, with wirelength as the objective function. We configure the floorplanner to make multiple tries until it satisfies the fixed-outline constraint. In our experiments, the floorplanner typically succeeded on the first try, but the ratio of successes to failures may depend on the amount of white-space in the design.

In our experiments the annealer found good floorplans where some macros were moved from their locations in the initial floorplan (see Figure 10(b)). We therefore believe that closely following the initial floorplan is not necessary for wirelength minimization and that the necessary information from the initial placement is captured by the physical clustering. However, if other design concerns encourage the preservation of macro placements, one could use more incremental force-directed macro placers [Mo et al. 2000]. Alternatively, one could tie those macros with faked wires to faked pins placed in strategic locations. We tried a variant of our flow in which we employed low-temperature annealing in the floorplanning stage in an attempt to preserve the initial locations of macros. The initial sequence pair for floorplanning is constructed from the placed shredded design. Low-temperature annealing is employed with slack-based moves to satisfy the fixed-outline constraints. Snapshots of different stages of the flow with low temperature annealing are illustrated in Figure 11. Note that the relative and absolute positions of macros in Figure 11(c) are close to the initial macro positions in Figure 11(a).

There are a number of factors and choices at the floorplanning stage that can affect the final placement. We list them below:

—*White-space available in the design*. Fixed-outline constraints can be easily satisfied for designs with large amount of white-space. However, for constrained designs the floorplanner can take a large amount of time in trying to satisfy the fixed-outline constraints.

—*Area assigned to the soft clustered blocks of standard cells*. The area assigned to the clustered block is the sum of the areas of the standard cells forming the cluster. However, for constrained designs with limited white-space, one can try to reduce the areas of these clustered macros to make it easier for the floorplanner to find a solution satisfying the outline constraints. However, floorplanning using sequence-pairs compacts blocks down and leftward. Therefore, if you have a large amount of white-space in the design, area-minimization will ensure that no macro is placed in the top-right corner. This may harm the solution quality if achieving minimum wirelength requires placing a particular macro in the top-right corner. Therefore, for large white-space designs reducing the area of the clustered block can hurt wirelength optimization.

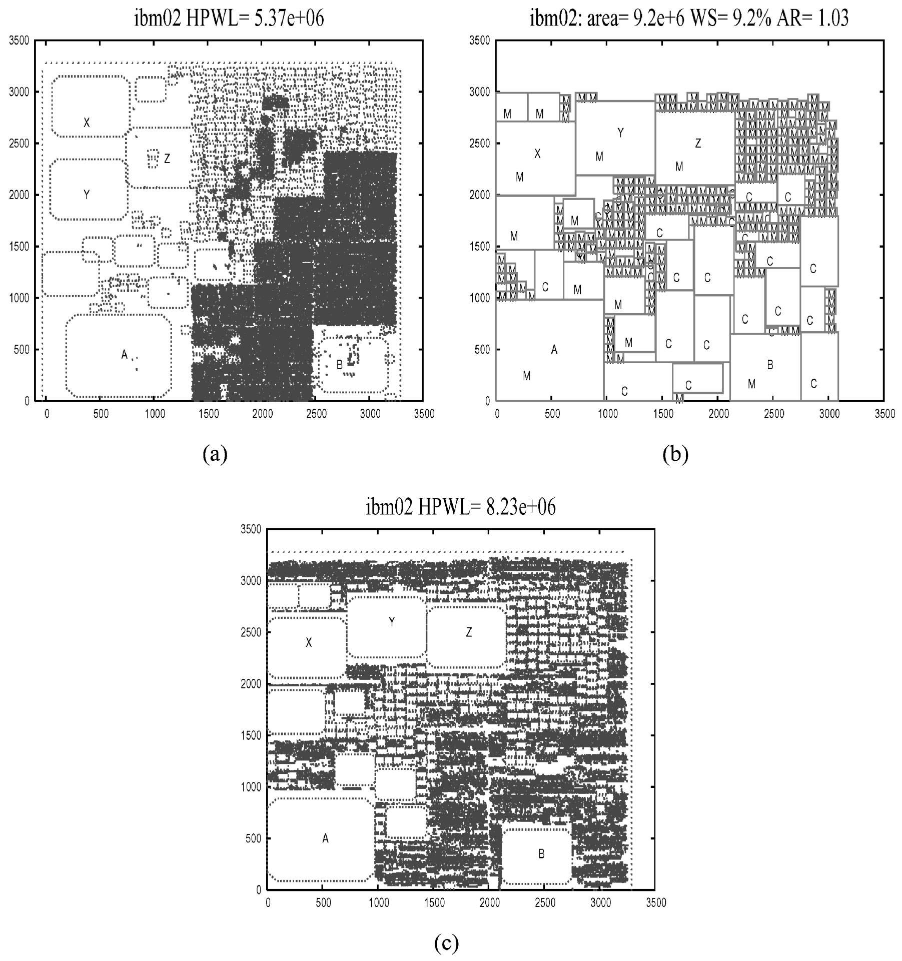


Fig. 11. Same mixed-size placement flow as that described in Figure 10. However, during the floorplanning stage (b), low-temperature annealing is used in an attempt to preserve the initial locations of macros obtained by placing the shredded netlist. The initial sequence-pair for floorplanning is constructed from the existing placement (a). As seen from (a), (b), and (c) positions of macros A, B, X, Y, Z are close to their original locations. However, runtimes for the floorplanning stage increase because of larger number of tries required to satisfy the fixed-outline constraints when using low-temperature annealing.

—*Trying to preserve the original locations of macros*. One might want to preserve the initial locations of the macros provided by the placement of the shredded netlist. In this case, the purpose of floorplanner is to only remove the overlaps.

We study the effect of these choices on the final placements in Section 5.2.1.

3.4 Final Standard-Cell Placement with Fixed Macros

The final locations of the macros are taken from successful fixed-outline floorplans, and the macros are fixed in the original layout. All cell sites below the macros are removed, and cell rows may need to be split into subrows. This enables the standard-cell placer to place the remaining movable standard cells without overlaps with the macros. In our experiments, we used the Capo mincut placer [Caldwell et al. 2000a], followed by two passes of window-based branch-and-bound placement.[2] Figure 10(c) shows the final placement generated by our proposed flow for the ibm02 design.

Min-cut placers that uniformly distribute white-space [Caldwell et al. 2000a, 2003] tend to produce excessive wirelength when large amounts of white-space are present [Alpert et al. 2002]. In our flow, during the final placement of standard cells around the macros, the white-space available in the designs (20% for ibm benchmarks) is distributed uniformly around the design with routability in mind. However, if the placement objective is only to minimize wirelength (as in this study), one can use more intelligent white-space allocation techniques [Alpert et al. 2002]. As a variant of our original flow, in the final stage of our flow we add unconnected filler cells to the design to represent the excessive white-space and reduce the white-space available to the placer to 10%. Thus the standard cells are placed more compactly, with the filler cells occupying vacant areas of the chip. The effect of filler cells on the final placement is studied in Section 5.2.1. However, we also point out that reducing HPWL may result in worsening the routability of a design.

4. MIXED-SIZE PLACEMENT FLOW 2

Our second proposed flow for mixed-size placement combines a black-box standard cell placer and a force-directed placer [Eisenmann and Johannes 1998]. The flow is described as follows.

4.1 Shredding Macro Cells

This step is identical to that in Section 3.1. The netlist is first preprocessed and all the macros in the design are decomposed into tightly connected smaller subcells of minimal height. For each macro the subcells are connected in a grid-like fashion. The shredded design with fake subcells and nets is placed using the Capo placer. The locations and orientation of the macros are determined from the placement of the shredded netlist. This initial placement can have overlaps between macros. The second step attempts to make the placement overlap-free. The first step entails placing a random logic netlist with embedded grid graphs. Standard cell placers Capo, Dragon, and Cadence QPlace have no difficulties in placing these netlists. However the force-directed placer Kraftwerk often fails to converge to a solution on such netlists.

4.2 Overlap Removal Using Force-Directed Techniques

We employ the ECO capabilities of the force-directed placer Kraftwerk [Eisenmann and Johannes 1998] to remove the overlaps from the initial placement while disturbing the initial placement as little as possible. As explained in

[2]As the detailed placement step, we apply branch-and-bound end-case placers [Caldwell et al. 2000b] using sliding windows.

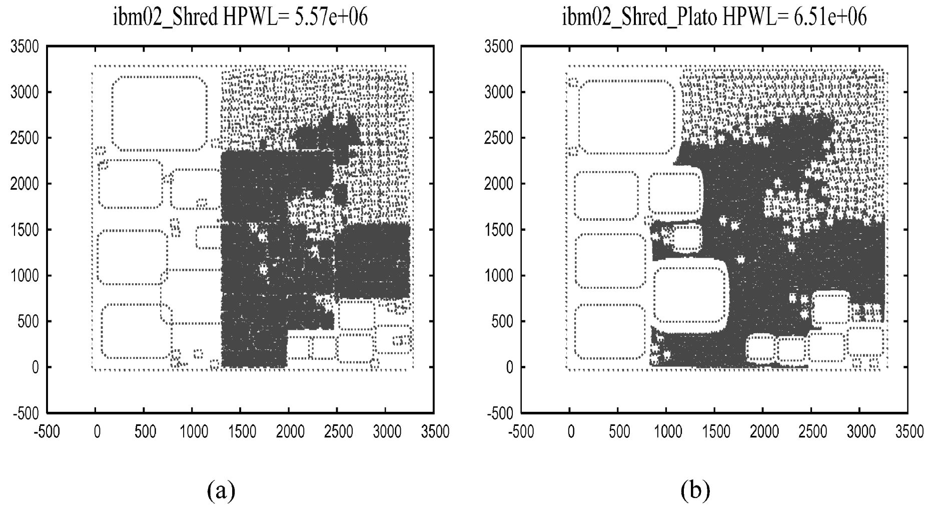


Fig. 12. Mixed-size placement Flow 2 as that described in Section 4. (a) The placement obtained after placing the shredded netlist. This is the same as in the first flow. (b) The overlap free placement obtained after running Kraftwerk in ECO mode on the placement in (a). Sometimes Kraftwerk is not successful in removing all the overlaps.

Section 2.2, Kraftwerk can take an initial placement and work in an ECO mode. In the ECO mode, Kraftwerk starts from the given placement and introduces additional forces according to density deviations arising because of the existing overlaps. The additional forces move the surroundings slightly in order to remove the overlaps. The algorithm tries to preserve the relative placement of cells. If the overlaps in the initial placements are small the additional forces are small resulting in small changes for the placement. However, if the overlaps in the initial placement are large this procedure can result in large changes to the initial placement and in some cases may also produce placement with large overlaps. Kraftwerk stops its placement iterations once the placement density in each region is below a certain threshold. However, this may result in small overlaps between the macros. From our experiments we conclude that the percentage of these overlaps with respect to the total layout area is fairly small. Alternatively one could employ other techniques [Mo et al. 2000; Nag and Chaudhary 1999; Vijayan 1991] to attempt and remove the overlaps. Figure 12 shows the placement obtained after running this flow on design ibm02. As seen, the final placement corresponds very accurately to the initial seed placement.

5. RESULTS

Our proposed flow is implemented in C++ and compiled by `g++ 2.95.4 -O3`. Runtimes are measured on a 2-GHz PC/Intel system running Linux. We compare our results against QPlace v.5.1.67 from Cadence, whose runtimes are measured on a 500-MHz Sun Blade-100 system running Solaris. We also compare our results against force-directed placer, Kraftwerk [Eisenmann and Johannes 1998], and mPG-MS [Chang et al. 2003]. Runtimes for Kraftwerk

Table III. Benchmark Characteristics (Column A_m^b shows the area of the largest macro in the design as % of the total cell area. Column ΣA_m shows the total area of the macros as % of the total cell area. Column $A_m^b : A_m^s : A_c^s$ shows the ratios of areas of the biggest macro to the smallest macro to the smallest standard-cell in the design.)

Circuit	#Nodes	#Nets	#Macros	A_m^b	ΣA_m	$A_m^b : A_m^s : A_c^s$
ibm01	12752	14111	246	6.37%	67.13%	8416:252:1
ibm02	19601	19584	280	**11.36%**	76.89%	30042:240:1
ibm03	23136	27401	290	**10.75%**	70.75%	33088:240:1
ibm04	27507	31970	608	**9.15%**	59.82%	26593:240:1
ibm05	29347	28446	0	N/A	N/A	N/A
ibm06	32498	34826	178	3.95%	72.90%	36347:175:1
ibm07	45926	48117	507	4.75%	52.56%	17578:240:1
ibm08	51309	50513	309	**12.10%**	67.35%	50880:240:1
ibm09	53395	60902	253	5.42%	52.42%	29707:240:1
ibm10	69429	75196	786	4.79%	81.37%	71299:252:1
ibm11	70558	81454	373	4.47%	49.76%	29707:240:1
ibm12	71076	77240	651	6.42%	73.00%	74256:152:1
ibm13	84199	99666	424	4.22%	47.64%	33088:240:1
ibm14	147605	152772	614	1.98%	26.72%	17860:144:1
ibm15	161570	186608	393	**10.99%**	43.34%	125562:240:1
ibm16	183484	190048	458	1.89%	48.71%	31093:252:1
ibm17	185495	189581	760	0.94%	23.78%	12441:252:1
ibm18	210613	201920	285	0.96%	11.96%	10152:243:1

are measured on a 2-GHz PC/Intel system and for mPG-MS are observed on a 750-MHz Sun Blade-1000 system running Solaris.

5.1 Benchmarks

The benchmarks used in our experiments are derived from the ISPD-98 (IBM) circuit benchmarks [Alpert 1998]. We converted the netlists into the Bookshelf placement format [Caldwell et al.], added placement-related information, and made the new benchmarks available on the Web.[3] The original descriptions specify cell areas, but not their dimensions. Since, in the `ibm` benchmarks, all areas are divisible by 16, we define rows of height 16. Cell sites in all rows have width 1. Cell widths were computed by dividing cell areas by row height (16). When the width of a cell exceeded a threshold number of sites (100 in our case), we upgraded such a cell to the status of a multirow macro with aspect ratio 1. The height of such a macro is computed by rounding the square-root of the area to the closest integer multiple of row height (16). The width is computed by dividing cell area by cell height and rounding the result to the closest integer number of cell sites. All designs have a white-space of 20% and their pads (marked in the original IBM netlists) were randomly placed near the perimeter of the core area. We converted the newly created benchmarks to the Cadence LEF/DEF format and applied Cadence's standard-cell placer QPlace to them.

5.2 Flow 1

Statistics for the new benchmarks are given in Table III, together with performance results of our Flow 1 with the Capo placer [Caldwell et al. 2000a] and

[3]at `http://vlsicad.eecs.umich.edu/BK/ISPD02bench/`.

Parquet fixed-outline floorplanner [Adya and Markov 2001].[4] We detail runtimes of each step in our proposed design flow. The performance of the industry placer QPlace is given in the same table for comparison. Our flow improves wirelength by 10–50% on most benchmarks.

The complexity of the problem increases with the number of macros and their relative size. According to Table IV, the benchmarks with relatively large macros (`ibm02`, `ibm03`, `ibm04`, `ibm08`, and `ibm15`) are difficult for QPlace.[5]

In our Flow 1 the bottleneck is the fixed-outline floorplanning stage, namely, in the wirelength computation that is performed independently for every move within the simulated annealing framework. While the number of nets in large netlists is typically proportional to the number of cells, many of those nets are not internal to physical clusters which serve as blocks during fixed-outline floorplanning. In other words, physical clustering reduces the number of movable objects much more than the number of nets. Since the relative white-space in the designs that we created was fairly small (20%), the fixed-outline floorplanner take more time to satisfy the fixed-outline constraints. For less constrained designs with more white-space, the run-times for the floorplanning stage can be significantly improved.

For benchmarks `ibm01`, `ibm17`, and `ibm18`, QPlace results are superior to our flow in terms of runtime. We believe that this is because the macros in these benchmarks are relatively small, and a standard-cell placer may handle them well enough. On the other hand, `ibm17` and `ibm18` are big enough to expose the coarseness of the 6×6 grid used in our experiments. Aside from increasing the grid size, it is possible to extend Capo to handle small macros, and thus entirely avoid running a floorplanner on those benchmarks.

5.2.1 *Sensitivities in Flow 1.* We study the various sensitivities in our flows. In our original Flow 1 the information about the initial locations of macros is not useful and placing the clustered netlist serves as a means to generate high-quality clusters. Also, fixed-outline floorplanning stage is a bottleneck in terms of runtime. The results in Table IV(B) are for the flow in which the floorplanning is done from scratch with a random initial solution and no attempt is made to preserve the initial positions of macros. We tried a variant of this flow in an attempt to maintain the initial macro locations obtained by placing the shredded netlist. We do this by forming a sequence-pair from the illegal placement obtained from step 1 of the flow and then employing low-temperature annealing. The results for this flow is presented in Table IV(C). For some benchmarks (`ibm02`, `ibm09`, and `ibm10`), the floorplanner requires more tries to satisfy the fixed-outline constraints because in the low-temperature annealing mode it is trying to massage an existing solution and the hill-climbing capabilities of simulated annealing do not work as efficiently. As a result the total runtimes for these designs increase. The final HPWL for most designs improve, but not significantly. We conclude that it is useful to preserve the initial positions of macros, especially in less area-constrained designs.

[4]The C++ source code of Parquet is available on the Web at `http://vlsicad.eecs.umich.edu/BK/parquet/`.
[5]We hope that extending QPlace with our proposed techniques can improve results for some circuits.

Table IV. Our Flow 1 (Capo + Parquet + Capo) Versus the Industry Placer QPlace v. 5.1.67. (Run-times for QPlace are measured on a 500-MHz Sun Blade 100 system with UltraSPARC-IIe processor; for Capo and Parquet they are measured on a 2 GHz Pentium Xeon. Parquet runtime includes all attempts to satisfy the given outline constraints; the number of attempts is shown as well. `ibm05` does not require a floorplanning stage (no macros). In Tables IV(C) and IV(D) we report results for Flow 1, with the floorplanner running in low-temperature annealing mode to preserve initial macro locations. Comparing with results from Table IV(B), we note that for some benchmarks the runtimes increase because of the larger number of attempts required to satisfy the fixed-outline constraints. The solution quality for most benchmarks improves with this flow. Table IV(C) shows results for the final placement of standard cells with uniform white-space distribution. Table IV(D) shows the results for final placement of standard cells along with filler cells for better white-space handling. Also, in Table IV(D), the areas of clustered macros of standard cells during floorplanning stage are reduced by 10% to make it easier for the floorplanner to satisfy the fixed-outline constraints. Our comparison to QPlace is mostly a sanity check because QPlace is not designed to solve mixed-size placement instances.)

	QPlace (ver. 5.1.67)		Flow 1 = Capo + Parquet + Capo (High-temperature annealing) %Final overlap = 0						Improved Flow 1 = Capo + Parquet + Capo (Low-temperature annealing) %Final overlap = 0					
	A		B (Uniform WS)						C (Uniform WS)			D (Filler Cells + Uniform WS)		
						FP								
Ckt	Final WL (e6)	Total time	Final WL (e6)	Total time	Shred place time	Time	# Tries	Final place time	Final WL (e6)	Total time	#FP Tries	Final WL (e6)	Total time	#FP Tries
ibm01	3.41	5m	3.66	15m	4m	10m	1	1m	3.36	13m	1	3.05	20m	4
ibm02	10.27	1hr13m	8.73	21m	8m	11m	1	2m	8.23	4hr0m	15	6.83	11m	1
ibm03	19.61	2hr23m	12.50	27m	7m	17m	1	3m	11.53	22m	1	10.38	59m	6
ibm04	37.31	6hr31m	12.99	31m	10m	18m	1	3m	11.93	25m	1	10.11	15m	1
ibm05	12.09	8m	11.20	5m	-	-	-	5m	11.20	5m	-	11.1	5m	-
ibm06	16.03	1hr16m	9.96	24m	10m	9m	1	5m	9.63	19m	1	9.94	18m	1
ibm07	22.04	1hr4m	16.37	47m	16m	23m	1	8m	15.80	39m	1	15.25	25m	1
ibm08	24.69	2hr36m	19.57	48m	16m	24m	1	8m	18.85	1hr51m	3	17.91	29m	1
ibm09	36.28	4hr10m	19.45	3hr23m	21m	2hr53m	5	9m	17.52	2hr58m	6	19.88	29m	1
ibm10	61.07	6hr21m	59.73	3hr21m	54m	2hr11m	1	16m	53.58	8hr10m	3	45.46	1hr56m	1
ibm11	42.73	3hr34m	29.43	1hr24m	28m	41m	1	15m	26.47	1hr9m	1	29.4	45m	1
ibm12	71.69	7hr5m	59.16	2hr31m	42m	1hr33m	1	16m	55.12	1hr59m	1	51.1	1hr35m	1
ibm13	59.80	5hr20m	36.08	3hr24m	35m	2hr31m	2	18m	33.56	1hr28m	1	37.73	53m	1
ibm14	52.06	1hr32m	58.49	3hr21m	53m	1hr52m	1	36m	52.67	5hr33m	2	50.26	2hr35m	1
ibm15	126.26	15hr49m	69.05	3hr51m	65m	1hr56m	1	50m	64.69	4hr24m	2	65.0	3hr15m	1
ibm16	216.16	44hr46m	93.45	4hr14m	1hr53m	1hr29m	1	52m	83.14	9hr40m	4	82.9	2hr42m	2
ibm17	89.13	1hr36m	98.23	6hr1m	1hr40m	3hr18m	1	1hr3m	91.50	4hr9m	1	89.17	3hr8m	1
ibm18	58.11	1hr32m	53.70	3hr0m	1hr13m	50m	1	57m	54.11	6hr37m	5	51.84	2hr7m	1

We try another variant of the low-temperature annealing flow in an attempt to reduce the floorplanning overhead. When forming soft clusters of standard cells using physical clustering, we reduce the area of the clustered soft block by 10%. Thus the area of the clustered block is 0.9 * (sum of areas of subcells). This helps the fixed-outline floorplanner to find a solution that satisfies fixed-outline constraints faster. However, reducing the area of the clustered cells might affect the optimization in some cases. Step 4 of the flow fixes the macro locations to the ones provided by the floorplanner and replaces standardcells around the macros. The standard cells are placed around the macros and the white-space in the design is allocated uniformly around the chip. However, we can improve the wirelength of the design by improved white-space allocation. We introduce unconnected filler cells in the design to represent the excessive white-space and reduce the white-space available to the placer to 10%. The design is then replaced with the macros being fixed. Thus the standard cells are placed more compactly, resulting in improved wirelength. The results for this variant of the flow are presented in Table IV(D). As seen, wirelength and runtimes improve for most designs.

5.3 Flow 2

Table V shows the results for our Flow 2 which places the shredded netlist using Capo to create an initial placement and then uses Kraftwerk in ECO mode to remove the overlaps while changing the initial seed placement as little as possible. For these results we also report the overlap remaining in the placement as a percent of the layout area. This is because the Flow 2 does not produce completely overlap-free placements. The results for this flow are on average better than our Flow 1, but Flow 1 is guaranteed to produce completely overlap-free placements. We also compare our results with placements generated by Kraftwerk [Eisenmann and Johannes 1998] from scratch. As seen in Table V, Kraftwerk frequently produces placements with large overlaps. Our proposed Flow 2 produces much better placements in terms of wirelength and overlaps than Kraftwerk run from scratch. We also compare our results to mPG [Chang et al. 2003] and the mixed-size placement flow [Choi and Bazargan 2003]. Note that the flow in Choi and Bazargan [2003] produces placements with large overlaps.

The problem with Flow 1 is that some of its steps ignore information produced by previous steps. The macro locations generated by placing the shredded netlist of step 1 are discarded. Also the soft macro locations obtained by the floorplanning stage are discarded in the final placement. Flow 2, which uses force-directed techniques to legalize placements obtained from step 1 overcomes this problem. A recent work [Khatkhate et al. 2004] studies legalization of such mixed-size placements with minimal movement from the original locations. However, the methods proposed in Khatkhate et al. [2004] produce a placement that is packed to the left side.

6. ONGOING WORK

We are currently working on tight integration of floorplanning and placement techniques [Adya et al. 2004a] to handle mixed-size designs. We have integrated

Table V. Results for Our Flow 2 (On average the results are better than for Flow 1 in Table IV. However some overlaps remain in the final placement. The % overlaps is shown. We compare our results with mPG, Kraftwerk, and a mixed-size placement flow. Runtimes for Flow 2(A) and Kraftwerk (C) are observed on a 2-GHz Linux/Pentium machine. Runtimes for mPG (B) are observed on a Sun Blade 1000 workstations running at 750 MHz. Runtimes for the mixed-size placement Flow (D) are observed on a 900-MHz Linux/Pentium machine.)

	Flow 2 = Capo + Kraftwerk ECO A			mPG [Chang et al. 2003] B		Kraftwerk [Eisenmann and Johannes 1998] C			[Choi and Bazargan 2003] D		
Ckt	Final WL(e6)	Total time	% Over-lap	Final WL(e6)	Total time	Final WL(e6)	Total time	% Over-lap	Final WL(e6)	Total time	% Over-lap
ibm01	2.92	5m	0.87	3.01	18m	3.01	2m	3.5	2.78	2m	6.31
ibm02	6.5	11m	0.19	7.42	32m	7.58	9m	6.4	6.64	3m	5.52
ibm03	9.63	14m	0.96	11.2	32m	11.4	10m	1.29	10.4	5m	7.62
ibm04	11.2	14m	0.47	10.5	42m	12.1	12m	1.31	11.7	6m	7.85
ibm05	11.2	5m	0.00	10.9	36m	12.9	3m	0.03	13.1	5m	1.72
ibm06	7.9	17m	0.41	9.2	45m	10.2	12m	0.9	9.89	9m	5.46
ibm07	13.6	55m	0.56	13.7	1hr8m	17.1	19m	0.75	16.2	14m	7.04
ibm08	17.2	22m	0.73	16.4	1hr22m	18.2	21m	7.43	18.0	15m	8.06
ibm09	17.8	31m	0.70	18.6	1hr24m	19.1	28m	0.49	20.2	16m	7.91
ibm10	47.5	1hr8m	2.1	43.6	2hr52m	51.5	35m	7.1	42.3	27m	4.29
ibm11	25.1	41m	0.67	26.5	1hr52m	26.6	36m	0.82	27.6	23m	9.69
ibm12	47.5	51m	0.38	44.3	1hr33m	52.6	43m	10.8	48.0	27m	7.06
ibm13	33.4	1hr8m	0.80	37.7	1hr31m	35.9	55m	0.69	35.4	33m	10.11
ibm14	47.9	1hr57m	0.43	43.5	4hr36m	47.4	1hr14m	0.4	—	—	—
ibm15	66.8	2hr2m	0.34	65.5	6hr25m	73.7	1hr33m	0.6	—	—	—
ibm16	86.7	2hr46m	0.45	72.4	7hr16m	82.4	1hr34m	13.1	—	—	—
ibm17	87.6	2hr16m	0.38	78.5	10hr6m	92.2	1hr47m	0.35	—	—	—
ibm18	57.2	2hr38m	0.08	50.7	7hr17m	54.9	1hr50m	0.09	—	—	—

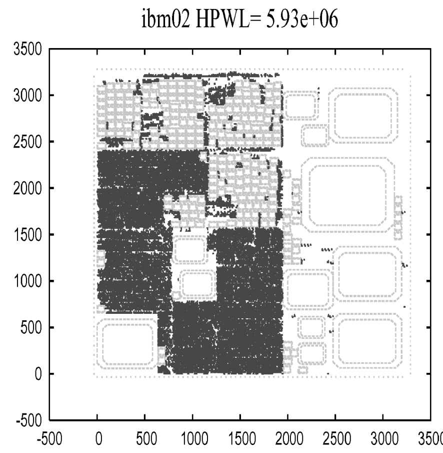


Fig. 13. ibm02 placement by our integrated placement and floorplanning flow. The macros are marked by double lines.

the fixed-outline floorplanner Parquet [Adya and Markov 2001, 2003] with the top-down placer Capo [Caldwell et al. 2000a] based on recursive bisection, seeking to improve scalability when handling mixed-size designs. Capo's framework is briefly described in Section 2.1. We modify this framework to handle mixed-size designs as follows. The large macros are initially treated as normal placeable cells and the placement block is processed in the regular fashion. Fixed-outline floor planning is employed to place the macros at legal locations inside the placement block if at least one of the following conditions is satisfied:

—The placement block has at least one large macro whose height/width is greater than a certain fraction (in our case $^1/_4$) of the block's height/width.
—The total area of the macros in the placement block is greater than a certain threshold (80% in our case) of the total cell area and the number of macros in the placement block is less than 100.

In order to use the floorplanner, a floorplanning instance is formed by clustering the standard cells with highest connectivity in a bottom-up fashion as explained in Section 3.2. The macros identified before are not clustered. The fixed-outline constraints are derived from the placement block's dimensions. If the fixed-outline floorplanning is successful, the macros are fixed at legal locations provided by the floorplanner, the sites are removed below the fixed macros, and the macros are removed from the block. From now on the placement block is processed as a normal placement block which has only standard cells. In theory, this proposed flow is a correct-by-construction approach and will produce a legal placement assuming that fixed-outline floorplanning succeeds in all cases. However, in our current implementation the fixed-outline floorplanning sometimes fails to find a legal solution satisfying the fixed-outline constraints. We believe that this is due to the difficulties in integrating recursive bisection and fixed-outline floorplanning. Figure 13 shows the placement of the ibm02 design obtained using this strategy. We present preliminary results for this flow in

Table VI. Results for Integrated Placement and Floorplanning Flow (We report HPWL, runtimes, and the final overlap between macros as a percentage of the total layout area. Runtimes are observed on a 2-GHz Linux/Pentium machine. Designs ibm08 and ibm10 take relatively longer times because of multiple floorplanning attempts to satisfy fixed-outline constraints during the placement flow.)

	Flow 3(Capo)	
Ckt	WL(e6)	Time
ibm01	2.96	1.8m
ibm02	5.93	4.6m
ibm03	9.74	6.8m
ibm04	10.6	8.0m
ibm05	11.0	4.6m
ibm06	7.35	7.1m
ibm07	12.5	17.4m
ibm08	15.6	2hr26m
ibm09	17.1	12m
ibm10	37.4	2hr13m
ibm11	23.6	16m
ibm12	44.5	58.8m
ibm13	31.0	22m
ibm14	44.5	38.5m
ibm15	61.1	56.9m
ibm16	70.5	60.2m
ibm17	80.6	45.9m
ibm18	50.9	44.6m

Table VI. Comparing with Tables IV and V, we see that, in most cases, this flow produces better HPWL placements and requires less runtime. In some cases final placements have overlaps. Since these overlaps tend to be extremely small, they can be removed by techniques from Khatkhate et al. [2004]. However, the Khatkhate et al. [2004] techniques require left-packing of the placement, which we would like to avoid [Adya et al. 2003a], because from our experience it is likely to cause routability problems. On the IBM mixed-size benchmarks, our proposed placement approach reliably produces overlap-free placements. The fact that Khatkhate et al. [2004] also uses the recursive partitioning approach and achieves lower wirelength than reported in this article suggests that more work is needed on mixed-size placement.

7. CONCLUSIONS

Modern SoC designs entail placement instances with numerous design IP blocks. Handling such layout problems has become important, and our work addresses this problem. Floorplanning techniques handle designs with macros, but do not scale to a hundred thousand standard cells. On the other hand, standard-cell placers handle large numbers of small, fixed-height cells, but do not handle macros very well. Therefore, we attempt to combine the strengths of both techniques.

We propose two design flows to place macro cells consistently with large numbers of standard cells. The first flow uses a combination of techniques from standard-cell placement and fixed-outline floorplanning. In particular, a number of existing placers can be used without source code modifications. Our proposed Flow 1 can be summarized as follows:

—Shred the original netlist. Use a standard-cell placer to generate an initial placement.
—Construct a floorplanning instance using a physical clustering algorithm.
—Generate valid locations of macros with an improved fixed-outline floorplanner.
—Fix the macros and place the remaining standard cells.

This flow can be modified to include a human designer who uses the initial placement as a hint when manually placing macros. Alternatively, variants of this flow can better preserve the initial placement.

Our proposed Flow 2 combines a standard cell placer with force-directed techniques and can be summarized as follows:

—Shred the original netlist. Use a standard-cell placer to generate an initial placement.
—Use a force-directed placer in ECO mode to remove the overlaps while trying to minimize the change in existing placement.

Our Flow 1 produces completely overlap-free placements with reasonably good wirelengths. Our Flow 2 produces high-quality wirelength placements with potentially some overlaps. However, these overlaps are generally very small and can be removed by simple techniques. Either of our flows can be applied for mixed-size design placement depending upon the requirements and characteristics of the design. Our empirical results for mixed-size placement are significantly better than those produced by the Cadence placer QPlace. Our results also compare favorably to those in Chang et al. [2003] and Choi and Bazargan [2003]. It should be noted, however, that the multilevel techniques in Chang et al. [2003] are very different from those used by other researchers and can, in principle, be combined with ours or even applied to placements produced by our methods.

Our experiments show that the proposed flows scale up to at least a thousand macros in addition to hundreds of thousands of standard cells. However, in Flow 1, floorplanning instances with a thousand blocks is a bottleneck and may be improved further. Our ongoing work focuses on techniques for incremental wirelength computation as well as multilevel techniques for floorplanning that can handle greater numbers of macros. We have not explicitly considered timing and congestion, but the significant improvements in wirelength obtained suggest that those metrics can also improve. Moreover, if an objective function can be quickly computed (e.g., circuit delay without false paths can be computed by static timing analysis in linear time), its optimization can be quickly added to simulated annealing that we use for floorplanning. Alternatively, one could use a previously reported force-directed macro block placer [Mo et al. 2000] that handles congestion. Congestion and timing can also be addressed at the second call to a black-box placer, assuming that the placer has relevant functionalities

[Wang and Sarrafzadeh 2000; Kahng et al. 2002]. Our focus on half-perimeter wirelength is also due to our belief that any large-scale layout tool that cannot successfully optimize wirelength is not going to successfully optimize more complex objectives. Our work can be considered a first step in this direction.

REFERENCES

Adya, S. N., Chaturvedi, S., Roy, J. A., Papa, D., and Markov, I. L. 2004a. Unification of partitioning, floorplanning and placement. In *Proceedings of the International Conference on Computer Aided Design (ICCAD)* (San Jose, CA). 550–557.

Adya, S. N. and Markov, I. L. 2001. Fixed-outline floorplanning through better local search. In *Proceedings of the International Conference on Computer Design* (Austin, TX). 328–334.

Adya, S. N. and Markov, I. L. 2002. Consistent placement of macro-blocks using floorplanning and standard-cell placement. In *Proceedings of the International Symposium on Physical Design* (San Diego, CA). 12–17.

Adya, S. N. and Markov, I. L. 2003. Fixed-outline floorplanning: Enabling hierarchical design. *IEEE Trans. VLSI Syst. 11*, 6 (Dec.), 1120–1135.

Adya, S. N., Markov, I. L., and Villarrubia, P. G. 2003a. On whitespace and stability in mixed-size placement and physical synthesis. In *Proceedings of the International Conference on Computer Aided Design* (San Jose, CA). 311–318.

Adya, S. N., Yildiz, M., Markov, I. L., Villarrubia, P. G., Parakh, P. N., and Madden, P. H. 2003b. Benchmarking for large-scale placement and beyond. In *Proceedings of the International Symposium on Physical Design* (Monterey, CA). 95–103.

Adya, S. N., Yildiz, M., Markov, I. L., Villarrubia, P. G., Parakh, P. N., and Madden, P. H. 2004b. Benchmarking for large-scale placement and beyond. *IEEE Trans. Comput. Aided. Des 23*, 4 (April), 472–487.

Alpert, C. J. 1998. The ISPD98 circuit benchmark suite. In *Proceedings of the International Symposium on Physical Design* (Monterey, CA). 80–85. Available online at `http://vlsicad.cs.ucla.edu/~cheese/ispd98.html`.

Alpert, C. J., Huang, J., and Kahng, A. B. 1997. Multilevel circuit partitioning. In *Proceedings of the Design Automation Conference* (Anaheim, CA). 530–533.

Alpert, C. J., Nam, G. J., and Villarrubia, P. G. 2002. Free space management for cut-based placement. In *Proceedings of the International Conference on Computer Aided Design* (San Jose, CA). 746–751.

Cadence. 2000. Openbook documentation for QPlace version 5.1.67. Cadence, San Jose, CA.

Caldwell, A. E., Kahng, A. B., and Markov, I. L. VLSI CAD Bookshelf. Available online at `http://vlsicad.eecs.umich.edu/BK`.

Caldwell, A. E., Kahng, A. B., and Markov, I. L. 2000a. Can recursive bisection alone produce routable placements? In *Proceedings of the Design Automation Conference* (Los Angeles, CA). 477–482.

Caldwell, A. E., Kahng, A. B., and Markov, I. L. 2000b. Optimal partitioners and end-case placers for standard-cell layout. *IEEE Trans. Comput.-Aided Des. 19*, 11, 1304–1314.

Caldwell, A. E., Kahng, A. B., and Markov, I. L. 2003. Hierarchical whitespace allocation in top-down placement. *IEEE Trans. Comput.-Aided Des. 22*, 11 (Nov.), 716–724.

Chang, C., Cong, J., and Yuan, X. 2003. Multi-level placement for large-scale mixed-size ic designs. In *Proceedings of the ASPDAC* (KitaKyushu, Japan). 325–330.

Choi, W. and Bazargan, K. 2003. Hierarchical global floorplacement using simulated annealing and network flow area migration. In *Proceedings of the DATE* (Munich, Germany). 1104–1105.

Dally, W. J. and Chang, A. 2000. The role of custom design in ASIC chips. In *Proceedings of the Design Automation Conference* (Los Angeles, CA). 643–647.

Doll, K., Johannes, F. M., and Antreich, K. J. 1994. Iterative placement improvement by network flow methods. *IEEE Trans. Comput.-Aided Des. 13*, 10 (Oct.), 1189–1200.

Dutt, S. 2000. Effective partition-driven placement with simultaneous level processing and a global net view. In *Proceedings of the International Conference on Computer Aided Design* (San Jose, CA). 254–259.

Eisenmann, H. and Johannes, F. M. 1998. Generic global placement and floorplanning. In *Proceedings of the Design Automation Conference* (San Francisco, CA). 269–274.

Fujuyoshi, K. and Murata, H. 1999. Arbitrary convex and concave rectilinear block packing using sequence pair. In *Proceedings of the International Symposium on Physical Design* (Monterey, CA). 103–110.

Hong, X., Huang, G., Cai, Y., Gu, J., Dong, S., Cheng, C.-K., and Gu, J. 2000. Corner Block List : An effective and efficient topological representation of non-slicing floorplan. In *Proceedings of the International Conference on Computer Aided Design* (San Jose, CA). 8–13.

Kahng, A. B. 2000. Classical floorplanning harmful? In *Proceedings of the International Symposium on Physical Design* (San Diego, CA). 207–213.

Kahng, A. B., Mantik, S., and Markov, I. L. 2002. Min-max placement for large-scale timing optimization. In *Proceedings of the International Symposium on Physical Design* (San Diego, CA). 143–148.

Karypis, G., Agarwal, R., Kumar, V., and Shekhar, S. 1997. Multilevel hypergraph partitioning: Applications in vlsi design. In *Proceedings of the Design Automation Conference* (Anaheim, CA). 526–529.

Khatkhate, A., Agnihotri, A. R., Yildiz, M. C., and Madden, P. H. 2004. Recursive bisection based mixed block placement. In *Proceedings of the International Symposium on Physical Design* (Phoenix, AZ). 84–89.

Lin, J. and Chang, Y. 2001. TCG: A transitive closure graph based representation for non-slicing floorplans. In *Proceedings of the Design Automation Conference* (Las Vegas, NV). 764–769.

Mo, F., Tabbara, A., and Brayton, R. K. 2000. A force-directed macro-cell placer. In *Proceedings of the International Conference on Computer Aided Design* (San Jose, CA). 404–407.

Murata, H., Fujiyoshi, K., Nakatake, S., and Kajitani, Y. 1996. VLSI module placement based on rectangle-packing by the sequence pair. *IEEE Trans. Comput.-Aided Des. 15*, 12, 1518–1524.

Murata, H. and Kuh, E. S. 1998. Sequence-pair based placement methods for hard/soft/preplaced modules. In *Proceedings of the International Symposium on Physical Design* (Monterey, CA). 167–172.

Nag, S. and Chaudhary, K. 1999. Post-placement residual-overlap removal with minimal movement. In *Proceedings of the DATE* (Munich, Germany). 581–586.

Pang, Y., Cheng, C., and Yoshimura, T. 2000. An enhanced perturbing algorithm for floorplan design using the o-tree representation. In *Proceedings of the International Symposium on Physical Design* (San Diego, CA). 168–173.

Sarrafzadeh, M., Wang, M., and Yang, X. 2002. *Modern Placement Techniques*. Kluwer, Dordrecht, The Netherlands.

Sherwani, N. 1999. *Algorithms for VLSI Physical Design Automation*. Kluwer, Dordrecht, The Netherlands.

Tang, X., Tian, R., and Wong, D. F. 2000. Fast evaluation of sequence pair in block placement by longest common subsequence computation. In *Proceedings of the DATE* (Paris, France). 106–111.

Tang, X. and Wong, D. F. 2001. FAST-SP: A fast algorithm for block placement based on sequence pair. In *Proceedings of the ASPDAC* (Yokohama, Japan). 521–526.

Varadrajan, R. and DeLendonck, G. 2002. Personal communication.

Vijayan, G. 1991. Overlap elimination in floorplans. In *Proceedings of the VLSI Design Conference.* 157–162.

Wang, X. Y. and Sarrafzadeh, M. 2000. Dragon2000: Standard-cell placement tool for large industry circuits. In *Proceedings of the International Conference on Computer Aided Design* (San Jose, CA). 260–263.

Yildiz, M. C. and Madden, P. H. 2001. Improved cut sequences for partitioning based placement. In *Proceedings of the Design Automation Conference* (Las Vegas, NV). 776–779.

Received May 2003; revised December 2003; accepted November 2003

RL-Huffman Encoding for Test Compression and Power Reduction in Scan Applications

MEHRDAD NOURANI and MOHAMMAD H. TEHRANIPOUR
The University of Texas at Dallas

This article mixes two encoding techniques to reduce test data volume, test pattern delivery time, and power dissipation in scan test applications. This is achieved by using run-length encoding followed by Huffman encoding. This combination is especially effective when the percentage of don't cares in a test set is high, which is a common case in today's large systems-on-chips (SoCs). Our analysis and experimental results confirm that achieving up to an 89% compression ratio and a 93% scan-in power reduction is possible for scan-testable circuits such as ISCAS89 benchmarks.

Categories and Subject Descriptors: B.7.3 [**Integrated Circuits**]: Reliability and Testing—*Test generation*; B.7.m [**Integrated Circuits**]: Miscellaneous; B.6.2 [**Logic Design**]: Reliability and Testing—*Test generation*

General Terms: Design, Algorithms

Additional Key Words and Phrases: Compression ratio, decompression, Huffman encoding, run-length encoding, scan-in test power, switching activities, test pattern compression, test compression, power reduction, scan applications

1. INTRODUCTION

Design for testability (DFT) based on scan and automatic test pattern generation (ATPG) is a reliable technique to achieve high fault coverage. Unfortunately, for large circuits the number of scan cells and test patterns becomes huge. This growth can be especially seen in today's systems-on-chips (SoCs). For large SoCs, test data volume increases, which in turn increases test cost due to the rise in test time and increased memory requirement.

Power dissipation is an important factor in today's chip design. Power dissipation in CMOS circuits is proportional to the switching activity in the circuit [Rabaey 2002]. During normal operation of a circuit, often a small number of flip flops change values in each clock cycle. However, during test operation,

This work was supported in part by the National Science Foundation CAREER Award #CCR-0130513.

Authors' addresses: Center for Integrated Circuits & Systems, Erik Jonsson School of Engineering and Computer Science, The University of Texas at Dallas, Box 830688, Richardson, TX 75083-0688; email: {nourani, mht021000}@utdallas.edu.

large numbers of flip flops switch, especially when test patterns are scanned into the scan chain. Compacting the test set often requires replacing (mapping) don't cares with specified bits "0" or "1" [Sankaralingam et al. 2000]. This process may increase switching activity of scan flip flops and eventually the scan-in power dissipation. There are usually plenty of don't cares in test patterns generated for scan, which provides an opportunity for compression and power reduction.

1.1 Prior Work

1.1.1 *Test Data Volume Reduction.* There are several techniques to address ATE-SoC interaction problems in terms of test data volume and application time. Built-in self-test (BIST) methodology reduces the need for expensive automatic test equipment (ATE) [Zorian et al. 1999]. In BIST, on-chip pseudorandom pattern generators and signature compressors are used. In practice, pseudorandom BIST cannot replace other test methods, because, especially for large chips, the existence of random pattern-resistant faults may lead to unacceptably long test times. To overcome these difficulties, deterministic test patterns need to be transferred from the ATE to the SoC under test. Several methods have been reported to reduce test volumes stored in the tester's memory [Iyengar et al. 1998; Das and Touba 2000; Hamzaoglu and Patel 2000; Hellebrand et al. 1995].

Compression techniques are used to speed up the ATE-SoC interaction during test. A data compression and decompression architecture for testing embedded cores in SoCs using Golomb coding [Golomb 1966] was presented in Chandra and Chakrabarty [2000]. A variable-length compression coding was presented in Chandra and Chakrabarty [2001] which carefully considers distributions of 0's in a test sequence. Chandra and Chakrabarty [2002b] addressed alternating run-length codes followed by frequency-directed run-length (FDR) coding for test data compression. A simultaneous reduction in volume of test data and power dissipation to generate minimum transition count (MTC) was presented in Rosinger et al. [2001, 2002]. The variable-length input Huffman compression (VIHC) method Gonciari et al. [2002] uses a maximum acceptable length to improve compression ratio, area overhead, and test application time. Jas et al. [1999] showed a compression/decompression based on Huffman coding of fixed-length blocks to reduce test data. The Illinois scan architecture provides a mechanism to reduce test application time and data storage requirements [Hsu et al. 2001]. An embedded deterministic test technology for a low-cost test that reduces scan test data volume and scan test time was presented in Rajski et al. [2002].

Reducing the number of scan chain input pins fed by the ATE using an on-chip decoder was proposed in Bayraktaroglu and Orailoglu [2001]. Reseeding the Linear Feedback Shift Register (LFSR) [Krishna et al. 2001] is another method that finds one or multiple seeds for an on-chip LFSR for every test vector. The generated bit sequence for LFSR matches the test vector at specified bit positions. The size of LFSR is significantly smaller than the scan chain. Reduction of scan test data in designs with multiple scan chains using a combinational decoder was presented in Reddy et al. [2002]. The basic approach proposed in [Khoche et al. 2002] is a fixed packet-based compression in which don't cares are filled appropriately in the test vectors such that they need not be stored in the ATE memory. Reordering test patterns is used to reduce the overall test application time [Wang and Chiou 2001; Tehranipour et al. 2003]. The LZ77 [Wolff and Papachristou 2002] and LZW [Knieser et al. 2003] methods use dictionary-based algorithms to compress the input test set with a large number of don't cares. A hybrid coding strategy, using a combination of run-length and dictionary-based methods, was proposed in Wurtenberger et al. [2003] to improve compression ratio.

1.1.2 *Scan Power Reduction.* The excessive switching activity during scan tests may cause larger peak and average power dissipation than those during normal operation. Therefore, the power constraint of the circuit needs to be considered in test mode. Applying scan test vectors generated by an ATPG is very time-consuming and thus compacting test vectors is unavoidable. Being a feature of the test set, overall power dissipation of cores remains the same. However, the compression/decompression procedure may intensively increase the power dissipation of scan elements in the chain. This is often called the *scan-in and scan-out power consumption* of the chain.

Several techniques have been proposed to reduce power consumption during test application. Test vector ordering [Girard et al. 1998], gated clock scheme [Bonhomme et al. 2001], scan latch partitioning [Nicolici and Al-Hashemi 2000], test generation for low-power scan testing [Wang and Gupta 1997], static compression to reduce power [Sankaralingam et al. 2000], mixed compression/decompression and low-power test application techniques based on Golomb codes [Chandra and Chakrabarty 2002a], FDR codes [Chandra and Chakrabarty 2002b], and MTC [Rosinger et al. 2001, 2002] are among the proposed techniques.

1.2 Main Contribution and Article Organization

In this article we present a compression/decompression technique to reduce test data volumes, test application times, and switching activities in scan cells. The proposed technique combines two well-known methods, run-length (RL) and Huffman encodings [Cover and Thomas 1991]. Essentially, RL encoding performs variable-length grouping to utilize the don't cares for (1) minimizing bit-transition and (2) compressing data by sending the length of running (similar) bits. Huffman encoding further enhances compression. While scan power reduction is not the main focus of this article, creating the minimum bit-transition by RL has a positive effect in reducing scan-in power consumption of the scan components. The compressed test data is scanned in and decompressed by an inexpensive on-chip decoder to generate the exact test pattern which is finally scanned into the scan chain.

Our work is similar to that of Chandra and Chakrabarty [2002b] and Rosinger et al. [2001] in using run-length encoding and to that of Gonciari et al. [2002] in using Huffman encoding. However, our technique is more flexible by (1) dealing with large runs of 0s or 1s effectively, (2) filling don't cares to maximize occurrence frequency and thus compression ratio and (3) being able to trade off between compression ratio and decoder cost.

This article is organized as follows. Section 2 describes the proposed compression technique. In Section 3, we explain the analytical foundation of our compression technique. The decompression method and architecture are discussed in Section 4. Power analysis is presented in Section 5. The experimental results are shown in Section 6. Finally, Section 7 concludes the article.

2. RL-HUFFMAN COMPRESSION TECHNIQUE

Typically, an ATPG generates test patterns in several steps [Synopsys, Inc. 2002]. First, it generates test patterns by pseudorandom pattern generation. When the fault coverage does not increase by generating more random patterns, it stops. The second step is to generate deterministic test patterns to detect all leftover random pattern-resistant faults to achieve higher fault coverage. There are often large numbers of don't cares in each test pattern. The ATPG may use don't care bits to compact (collapse) test vectors. Random bits are sometimes assigned to don't cares in test patterns to detect nonmodeled faults. In most computer-aided design (CAD) tools, the user has options for filling the don't cares. For example, in Synopsys's Tetramax [Synopsys, Inc. 2002], the *"norandomfill"* option can be used to leave don't cares unchanged in the generation phase.

2.1 Step 1: Run-Length Encoding

In today's SoCs, the percentage of don't cares can be quite high, sometimes more than 80% for large circuits [Wolff and Papachristou 2002]. IBM reported 98% of bits in test patterns for some of their designs were filled with don't cares [Koenmann 2000]. In our compression technique we take advantage of the presence of don't cares in test vectors. Our technique uses a variable encoding of test vectors. The basic idea in RL encoding is a careful replacement of don't cares with "0" or "1" to find the minimum number of $0 \rightarrow 1$ and $1 \rightarrow 0$ transitions. Minimizing such transitions also has a significant impact in reducing the dynamic switching activities of test components (i.e., scan cells) to which the vectors are sent.

Figure 1 shows an example of filling don't care bits with "0" or "1" based on RL strategy. As shown, the filling process starts from the most significant bit of the vector sequence (e.g., T_1). The don't cares are filled with 1s until we reach a "0". In the next step, don't cares are filled with 0s until the next "1". This process continues until it replaces all don't care bits with "0" or "1". In this example, the number of transitions is 10 and the test set is partitioned into 11 blocks ($N_b = 11$), each of which is filled with only 1s or 0s. L_i shows the length of blocks in the test set. The corresponding block lengths are shown in Figure 1(b). Block lengths are shown using $L_i[v]$ notation, where L_i is block length and $[v]$ is the bit value "0" or "1" that is filled in that block. The characters are stored in the lookup table for decoding purpose. As shown, the number of characters (N_l) stored in the lookup table is seven. Figure 1(c) summarizes the occurrence frequency (f_i) of these characters. In this example, no limit on the block size is

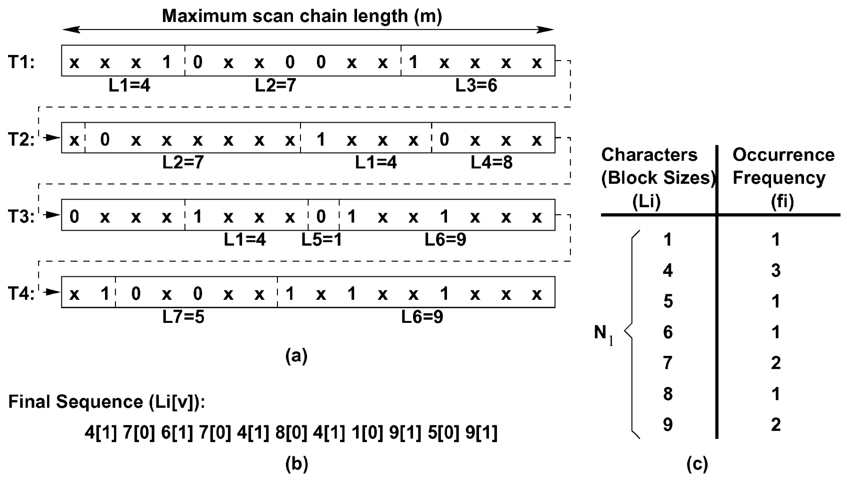

Maximum scan chain length (m)
T1: x x x 1 0 x x 0 0 x x 1 x x x x
L1=4 L2=7 L3=6
T2: x 0 x x x x x x 1 x x x 0 x x x
L2=7 L1=4 L4=8
T3: 0 x x x 1 x x x 0 1 x x 1 x x x
L1=4 L5=1 L6=9
T4: x 1 0 x 0 x x 1 x 1 x x 1 x x x
L7=5 L6=9
(a)
Final Sequence (Li[v]):
4[1] 7[0] 6[1] 7[0] 4[1] 8[0] 4[1] 1[0] 9[1] 5[0] 9[1]
(b)
Characters (Block Sizes) (Li)
Occurrence Frequency (fi)
1 1
4 3
5 1
6 1
7 2
8 1
9 2
N_l
(c)

Fig. 1. Applying the run-length algorithm to a small example.

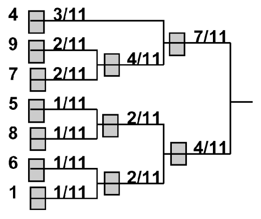
```
RL Encoding()
Input: Test set {T}
Output: $L_i$ and $f_i$, $i \geq 1$

01:$T_0$=First specified bit;
02:$i$=1;
03:for ($j$=1 to $n \cdot m$)
04:{
05:    if ($T_j = T_0$ or $T_j$ ='$x$')
06:        $i$++;
07:    else
08:    {
09:        $L_i = i$;
10:        $f_i$++;
11:        $T_0=\overline{T_0}$;
12:        $i$=1;
13:    }
14:}
15:$L_i = i$;
16:$f_i$++;
```

Fig. 2. RL encoding algorithm.

defined and we simply replace don't cares with 0/1 values to maximally enlarge a block. More sophisticated approaches can be devised for other objectives such as minimizing N_l or increasing f_i values. Some techniques are discussed in Section 3.4.

Figure 2 shows RL encoding algorithm, where n and m are the number of test patterns and the scan chain size, respectively. Again, we did not define any limit on the block size. In the next section we will elaborate further on the role of maximum block size. In the RL algorithm, T_j refers to the jth bit in a test sequence (combined patterns) when it is checked to form the blocks. In our method, within each block we have only 0s or 1s. If RL is the only compression step, instead of scanning actual data we can scan the length of



(a)

	Characters (Li)	Occurrence Frequency (fi)	Huffman Code (Ci)	Saving (Si)
N_l	4	3	00	+6
	9	2	010	+12
	7	2	011	+8
	5	1	100	+2
	8	1	101	+5
	6	1	110	+3
	1	1	111	-2
				S=+34

(b)

Fig. 3. Huffman code for the example in Figure 1.

that block showing how many bits in the block are “0” or “1.” Most of the other techniques focus on probability of the occurrence of fixed-length blocks which may include “0,” “1,” or “x” (don’t care) [Jas et al. 1999; Khoche et al. 2002]. In our case, the length of blocks (L_i values in Figure 1) are encoded in the first step of compression. Note carefully that, as lines 9–12 in Figure 2 indicate, the algorithm always generates alternating 0-blocks and 1-blocks. Lines 15–16 store the final block size and frequency in a test set. Therefore, a single bit T_0 to start the alternation is sufficient and the values of each block (e.g., [v] in Figure 1(b)) are not explicitly sent.

2.2 Step 2: Huffman Encoding

To get the best compression rate in the second step, we also apply Huffman coding (a variable-length encoding by nature) [Huffman 1952] to encode block length values (characters). The idea is to assign a smaller number of bits to codewords that occur most frequently and a larger number of bits to those that occur less frequently. Huffman codes are obtained by constructing a Huffman tree. Figure 3(a) shows the Huffman tree for the example of Figure 1(c) with the occurrence frequency annotated. The generated Huffman code is shown in Figure 3(b). Clearly, blocks with higher-occurrence frequencies will get shorter codewords. For this example, instead of sending 64 bits of original data, the ATE sends only $64 - 34 = 30$ bits. This is a 53.12% overall saving in test data volume. The saving in transfer time depends on working frequencies of the ATE and SoC under test. In Section 4 we will express the upper bound of transfer time saving based on compression saving. The last column in Figure 3(b) shows saving of data sent into the chip. Instead of sending L bit 0s or 1s, the l bit codeword is sent. Saving for the ith codeword is $L_i - l_i$, where l_i is the length of the ith codeword (C_i). Saving for sending the ith codeword with frequency f_i is $S_i = f_i(L_i - l_i)$. Total saving for all codewords is $S = \sum_{i=1}^{N_l} f_i(L_i - l_i)$. Then, the compression ratio (percentage of data reduction) for n vectors sent to a scan chain of m cells (overall $n \cdot m$ bits) will be

$$CR = \frac{S}{n \cdot m} = \frac{\sum_{i=1}^{N_l} f_i(L_i - l_i)}{\sum_{i=1}^{N_b} L_i}.$$

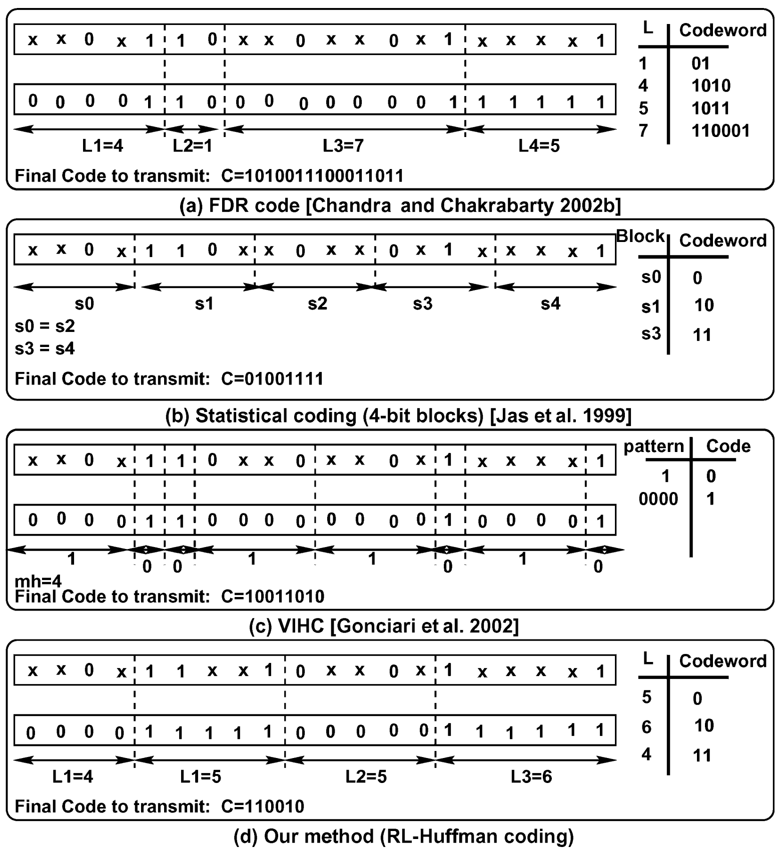

x x 0 x 1 1 0 x x 0 x x 0 x 1 x x x x 1
0 0 0 0 1 1 0 0 0 0 0 0 0 0 1 1 1 1 1 1
L1=4 L2=1 L3=7 L4=5
L | Codeword
1 | 01
4 | 1010
5 | 1011
7 | 110001
Final Code to transmit: C=1010011100011011
(a) FDR code [Chandra and Chakrabarty 2002b]
x x 0 x 1 1 0 x x 0 x x 0 x 1 x x x x 1
s0 s1 s2 s3 s4
s0 = s2
s3 = s4
Block | Codeword
s0 | 0
s1 | 10
s3 | 11
Final Code to transmit: C=01001111
(b) Statistical coding (4-bit blocks) [Jas et al. 1999]
x x 0 x 1 1 0 x x 0 x x 0 x 1 x x x x 1
0 0 0 0 1 1 0 0 0 0 0 0 0 0 1 0 0 0 0 1
1 0 0 1 1 0 1 0
mh=4
pattern | Code
1 | 0
0000 | 1
Final Code to transmit: C=10011010
(c) VIHC [Gonciari et al. 2002]
x x 0 x 1 1 x x 1 0 x x 0 x 1 x x x x 1
0 0 0 0 1 1 1 1 1 0 0 0 0 0 1 1 1 1 1 1
L1=4 L1=5 L2=5 L3=6
L | Codeword
5 | 0
6 | 10
4 | 11
Final Code to transmit: C=110010
(d) Our method (RL-Huffman coding)

Fig. 4. Comparison between several techniques.

Eventually, the RL-Huffman code is scanned into the chip. An on-chip decoder decompresses the encoded bits to recognize how many 0s or 1s need to be shifted into the scan chain. In general, the Huffman tree is needed not only for encoding but also for decoding. To make the decoder independent of the test set, the information of this tree can be transmitted with the compressed test data. However, this approach may result in higher cost and lower compression ratio.

Our decompression technique is quite straightforward and will be discussed in detail in Section 4. Also, in Section 5 we will show the positive impact of RL encoding on minimizing the bit transitions, and, hence, power consumptions of the scan cells. Note that Huffman encoding provides a further reduction and does not deteriorate the minimum transition property achieved by RL encoding.

2.3 Comparison to Other Techniques

To clarify why RL-Huffman encoding is so efficient, we compare our technique with techniques using FDR codes [Chandra and Chakrabarty 2002b], statistical coding [Jas et al. 1999] and VIHC [Gonciari et al. 2002]. Figure 4 shows a comparison between these techniques for one 20-bit test pattern.

In FDR codes, don't care bits are replaced with 0s and 1s. The blocks may have different lengths but they use a fixed code to make the decoder independent of test set. Such replacement and code reduce the cost of a decoder [Chandra and Chakrabarty 2002b] but the highest compression ratio may not be achieved (see Figure 4(a)). The statistical coding method proposed in Jas et al. [1999] uses a fixed-length coding combined with Huffman coding based on the occurrence frequency of codewords. Figure 4(b) shows the result of this method for 4-bit lengths. But the chance of having identical fixed-length blocks (and thus occurrence frequency) will be decreased when the size of blocks increases. In the VIHC method [Gonciari et al. 2002], a maximum acceptable length of runs of 0s is defined ($m_h = 4$ in Figure 4(c)). The test vector is divided into runs of 0s of lengths smaller or equal to m_h and then coded using Huffman encoding.

The final compressed data in our technique is shorter than other techniques, as shown in Figure 4(d). Note carefully that in the conventional RL coding we need to send both the characters and their lengths, which means 9[1]5[0]6[1] (i.e., nine 1s, five 0s, and six 1s) should be sent for the pattern in Figure 4(d). However, in our method we guarantee alternating transitions between a 0-block and a 1-block and therefore the values of "0" and "1" do not explicitly need to be sent along with the code. In Section 4 we will explain how a decoder takes this property into account and easily generates this alternation. Our method is similar to the method presented in Gonciari et al. [2002] in using a combination of RL and Huffman encoding. However, the application techniques are different. Specifically, in dealing with large block sizes, being able to target the occurrence frequency to maximize compression and tradeoff between the compression ratio and the decoder cost differentiates our method from similar techniques.

We acknowledge that comparing methods with only one pattern may not be fair as the average compression ratio for large test sets matters. This example is presented to differentiate our approach from others. In Section 6 we will show that the compression ratio of our method is also quite good for large test sets.

3. ANALYSIS OF RL-HUFFMAN COMPRESSION TECHNIQUE

RL encoding takes advantage of don't cares by creating blocks of identical data values. The Huffman code is a variable-length code that can do much better than a fixed-length code in terms of compression Cover and Thomas [1991]. Huffman coding can be done in *O(nlogn)* and therefore is not very time-consuming [Cormen et al. 2001]. Additionally, no codeword is prefix to the others, a property that makes decoding easy and inexpensive. Huffman coding gives frequent and infrequent characters short and long codewords, respectively.

3.1 Compression Efficiency

The lower bound of the average length of a codeword, used for encoding an information source, can be expressed by its *entropy* [Cover and Thomas 1991].

The entropy of a test set with N_l unique characters is given by

$$H = -\sum_{i=1}^{N_l} (p_i \cdot log_2 p_i),$$

where p_i is the occurrence probability of character L_i. As the above formula shows, entropy is independent of character lengths. The average length of a codeword is given by

$$l_{avg} = \sum_{i=1}^{N_l} p_i \cdot l_i,$$

where l_i is the codeword length for the character L_i. For a general compression technique we have $l_{avg} \geq H$, and efficiency of the compression technique is expressed as

$$E = l_{avg}/H.$$

The closer E is to 1, the more efficient is the compression technique. The average length of an Huffman codeword is the closest to the entropy of a test set [Cover and Thomas 1991]. In the example of Figure 1, $H = 2.663$, $l_{avg} = 2.727$, and $E = 1.023$.

3.2 Compression Ratio

The overall compression ratio depends on the saving that RL (S_{RL}) and Huffman (S_H) encodings achieve in the first and second step, respectively. Briefly, we use this formula in computing the overall compression ratio (CR):

$$CR = \frac{S_{RL} + S_H}{\sum_{j=1}^{N_b} L_j}.$$

More specifically, the compression ratio achieved in the first step (RL) will be $CR_{RL} = S_{RL}/\sum_{j=1}^{N_b} L_j$. The compression ratio in the second step (Huffman) with respect to the result of the first step can be computed as $CR_H = S_H/(\sum_{j=1}^{N_b} L_j - S_{RL})$. S_{RL} metric depends on the number and distribution of x's and also on the maximum block size (K). Briefly, in the RL step, instead of K-bit data, we send only $\lceil log_2 K \rceil$ bits. Therefore, if the maximum block size (maximum Run-Length) is K and there are N_b characters to transmit:

$$S_{RL} = \sum_{j=1}^{N_b} L_j - (\lceil log_2 K \rceil \times N_b),$$

where $\sum_{j=1}^{N_b} L_j = n \cdot m$. In the above formula N_b bits need to be added to $\lceil log_2 K \rceil$ if "1" and "0" (values for each block) need to be sent. In our method we don't include these values because alternating 0-blocks and 1-blocks are guaranteed. For the example of Figure 1(b) we have $S_{RL} = 64 - 4 \times 11 = 20$.

Since we use Huffman coding in the second step, the compression ratio of this step depends on the occurrence frequency of characters. The characters in our method are L_i values. Total length (bits) of data will be $B = \sum_{i=1}^{N_l} (f_i \times l_i)$, where N_l is the number of distinct characters (different L_i

values), f_i and l_i are the occurrence frequency and length of codeword C_i, respectively.

Using fixed-length binary coding, we have $l_i = \lceil log_2 N_l \rceil \; \forall i$. Therefore, for a total number of characters in the data set $N_b = \sum_{i=1}^{N_l} f_i$ we have

$$B_{\textit{fixed-length}} = \lceil log_2 N_l \rceil \times \sum_{i=1}^{N_l} f_i = \lceil log_2 N_l \rceil \times N_b.$$

It has been shown that, for Huffman coding, the average length of codewords (l_{avg}) satisfies this inequality: $H \leq l_{avg} \leq H + 1$, where $H = -\sum_{i=1}^{N_l}[(f_i/N_b) \times log_2(f_i/N_b)]$ [Cover and Thomas 1991]. Note that f_i/N_b shows the occurrence probability of codeword C_i in the data set. Therefore:

$$B_{\textit{Huffman}} \leq \left\{1 - \sum_{i=1}^{N_l}[(f_i/N_b) \times log_2(f_i/N_b)]\right\} \times N_b.$$

The upper bound of the Huffman compression ratio, compared to the fixed-length coding, will be

$$\begin{aligned} CR_H &\leq \frac{B_{\textit{fixed-length}} - B_{\textit{Huffman}}}{B_{\textit{fixed-length}}} \\ &\leq \frac{\lceil log_2 N_l \rceil - \{1 - \sum_{i=1}^{N_l}[(f_i/N_b) \times log_2(f_i/N_b)]\}}{\lceil log_2 N_l \rceil}. \end{aligned}$$

Researchers have shown that CR_H is in the range of 20% to 90% depending on the occurrence frequencies [Cormen et al. 2001]. In our example of Figure 3: $S_H = 4 \times 11 - 30 = 14$ and overall $CR = \frac{S_{RL}+S_H}{\sum_{j=1}^{N_b} L_j} = \frac{20+14}{64} = 53.12\%$.

3.3 Maximum Run-Length (K)

The overall compression ratio depends on distribution of don't cares and occurrence frequencies of characters to be transmitted. Additionally, the maximum block size (K) affects number of characters (N_l), number of blocks (N_b), and indirectly occurrence frequencies (f_i). More specifically, in our approach we need to choose K such that a reasonable probability of having a block of that size exists in a test data set. Very small and very large values for K both hurt the compression ratio. Intuitively, if we choose a very large K, the probability of having that many 0s or x's (or that many 1s and x's) running through K consecutive bits may be very small and thus we will not get high frequencies. On the other hand, if we choose K to be very small, the distribution of frequencies will be close to uniform and we will not get that much compression.

Note that there is no magic number for K. K can be chosen more efficiently based on the amount of don't cares in test data set. Suppose, $p(0)$, $p(1)$, and $p(x)$ show the probability of being "0," "1," and "x" when one bit is chosen from the data set, respectively. Obviously, these probabilities depend on the total number of 0s, 1s, and x's in a set and can be computed a priori. Let's assume $p(x) = D$ and $p(0) = p(1) = (1 - D)/2$, where D is the ratio of don't care bits to the total bits in a test set. The probability that a K-bit block is formed can be

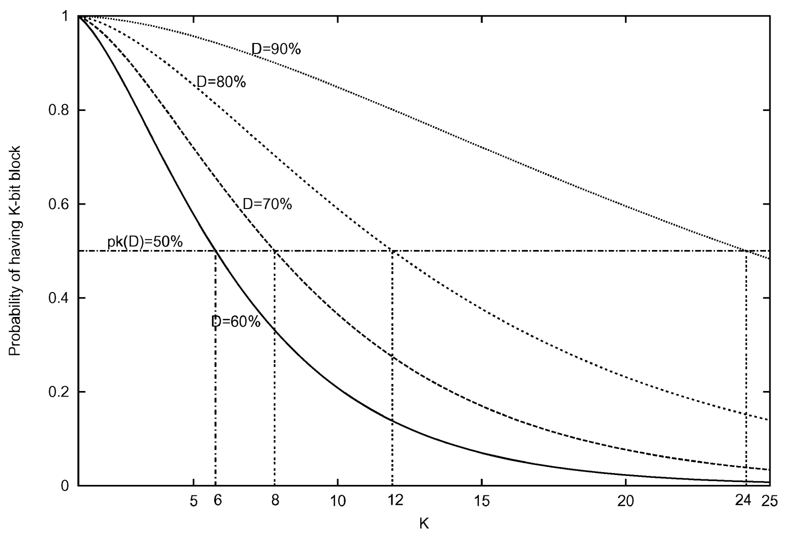


Fig. 5. Probability of having K-bit block (p_K) as a function of $D\%$.

computed [Tehranipour and Nourani 2002]:

$$\begin{aligned}
p_K(D) &= p(Kbits \in \{0,x\}) + p(Kbits \in \{1,x\}) \\
&\quad -p(Kbits \in \{x\}) \\
&= [p(0)+p(x)]^k + [p(1)+p(x)]^k - [p(x)]^k \\
&= 2^{1-K}(1+D)^K - D^K .
\end{aligned}$$

In Figure 5, $p_K(D)$ is drawn for various $D\%$ (don't care percentage) values between 60% to 90%. As K increases, the probability of having such block will be smaller. Due to the complicated relationship of factors and unavailability of frequencies, no magical K can work for all data sets. However, for a given data set, the ratio of don't care D values can be computed and an appropriate K can be chosen such that the probability remains reasonable. A large value of K hurts RL encoding because, for a fixed D, $p_K(D)$ becomes small. But it helps Huffman encoding as a large K implies larger saving. Similarly, a small value of K makes Huffman encoding ineffective because the block size distribution becomes almost uniform. But it helps RL encoding as $p_K(D)$ will be higher. Optimizing K is beyond the scope of this article. Empirically, we observed that choosing K such that the probability of having a block size of K (or very close to K) remains in the range of 0.4 to 0.6 produces a good compromise between the compression ratio and the decoder cost. In such a situation, *CR%* is often upper-bounded by $D\%$. Some experimental evidence will be shown in Section 6.

3.4 Techniques to Trade Off Compression Ratio and Decoder Cost

When RL encoding is applied, the length of blocks may vary from 1 to $n \cdot m$. Therefore, the decoder needs to decode all codewords between 1 and $n \cdot m$. For large circuits, the scan chain is too long, and may even be around a couple of hundreds. In such cases, the occurrence frequencies of f_i may not be too high because it is distributed from 1 to $n \cdot m$ and all block lengths between 1 to $n \cdot m$ may happen. The advantage of a long block is that a large amount of data is sent by a very small codeword. The disadvantage is that it needs a more costly decoder. In our technique, we use the maximum run length (K) to trade off complexity of the decoder and compression ratio.

We have investigated three techniques of which the first two limit block sizes to a predefined number K and the third technique has no limit on block size. The compression algorithm, shown in Figure 2, needs to be slightly revised to limit block sizes to a predefined value K. Our experimentation on tradeoff between K, compression ratio, and decoder cost will be reported in Section 6.

3.4.1 *Technique* 1. We have the following:

$$\forall L_i : L_i > K \Rightarrow L^*_{i1} + L^*_{i2} = L_i \ni \begin{cases} L^*_{i1} = K, \\ L^*_{i2} = L_i - K. \end{cases}$$

In Technique 1 if the size of a block (L_i) is greater than K, it is divided into two new blocks (with the size of K and $L_i - K$) and the process continues until no block size is greater than K. A 0 (blank) character is inserted between the new blocks. Obviously, zero length never happens for actual data blocks in our technique. The 0 character is used to guarantee alternation between 0s and 1s for consecutive blocks. This feature reduces the decoder cost with little sacrifice of the compression ratio.

3.4.2 *Technique* 2. We have the following:

$$\forall L_i : L_i > K \Rightarrow L^*_{i1} + L^*_{i2} = L_i \ni$$

$$S^*_{i1} + S^*_{i2} = \max_{L^*_{i1} + L^*_{i2} = L_i} \{S_{i1} + S_{i2}\}.$$

Technique 2 revisits the blocks and, if a block size is larger than a given K, decomposes it such that the break increases the occurrence frequencies of other characters to achieve maximum saving. Blank character (0) is still required to be added between each two blocks to ensure alternating 0-blocks and 1-blocks. This technique keeps the size of the decoder proportional to K, but may enhance the compression ratio.

3.4.3 *Technique* 3. We have the following:

$$\forall L_i : L_i > K \Rightarrow L^*_{i-1} + L^*_i + L^*_{i+1} = L_{i-1} + L_i + L_{i+1} \ni$$

$$S^*_{i-1} + S^*_i + S^*_{i+1} = \max_{L^*_{i-1} + L^*_i + L^*_{i+1} = L_{i-1} + L_i + L_{i+1}} \{S_{i-1} + S_i + S_{i+1}\}.$$

In Technique 3, blocks generated by the RL encoding algorithm are revisited to find the minimum number of blocks. Reducing the number of blocks reduces the entropy of a test set, which eventually increases saving and the compression ratio. Moreover, the cost of a decoder is reduced since a fewer number of states need to be handled by the decoder. No blank character needs to be added to the final sequence. This technique uses a graph-based heuristic for entropy optimization and is more complicated than Techniques 1 and 2. Details of these

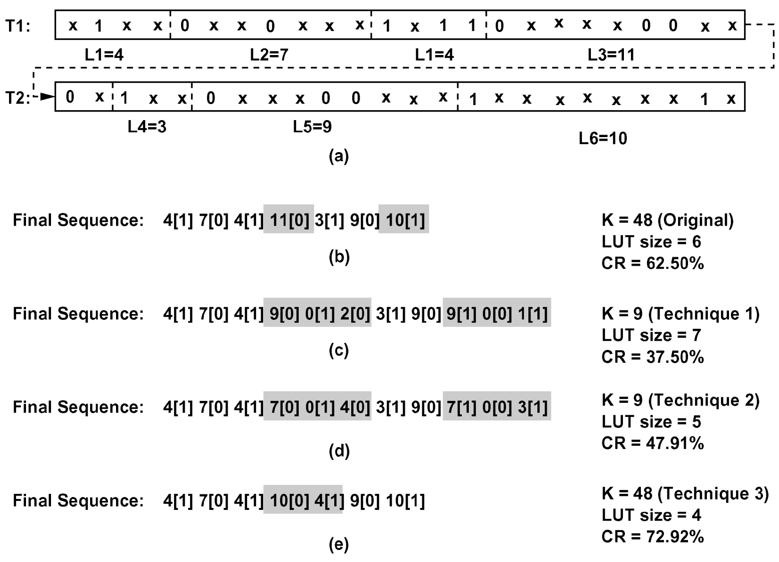

T1: x 1 x x 0 x x 0 x x x 1 x 1 1 0 x x x x 0 0 x x
L1=4
L2=7
L1=4
L3=11
T2: 0 x 1 x x 0 x x x 0 0 x x x 1 x x x x x x x 1 x
L4=3
L5=9
L6=10
(a)
Final Sequence: 4[1] 7[0] 4[1] 11[0] 3[1] 9[0] 10[1]
(b)
K = 48 (Original)
LUT size = 6
CR = 62.50%
Final Sequence: 4[1] 7[0] 4[1] 9[0] 0[1] 2[0] 3[1] 9[0] 9[1] 0[0] 1[1]
(c)
K = 9 (Technique 1)
LUT size = 7
CR = 37.50%
Final Sequence: 4[1] 7[0] 4[1] 7[0] 0[1] 4[0] 3[1] 9[0] 7[1] 0[0] 3[1]
(d)
K = 9 (Technique 2)
LUT size = 5
CR = 47.91%
Final Sequence: 4[1] 7[0] 4[1] 10[0] 4[1] 9[0] 10[1]
(e)
K = 48 (Technique 3)
LUT size = 4
CR = 72.92%

Fig. 6. Techniques to trade off compression ratio and decoder cost.

techniques are beyond the scope of this article and can be found in Tehranipour and Nourani [2002].

Figure 6 shows these three techniques applied to a small example. Figure 6(a) is the original RL-Huffman with no limit on K and no block size adjustment. Figure 6(b) shows the length of blocks computed with $K = n \cdot m = 48$ using the original RL technique. In this case, the lookup table size, which indicates decoder cost, is proportional to 6 entries and the compression ratio is 62.50%. Assuming $K = 9$, Figure 6(c) shows the first technique. Length 11[0] is decomposed to 9[0]0[1]2[0] and length 10[1] is decomposed to 9[1]0[0]1[1]. The lookup table size is 7 and the compression ratio is 37.50%.

Figure 6(d) shows the second technique for $K = 9$. The length 11[0] is decomposed to 7[0]0[1]4[0] and 10[1] is decomposed to 7[1]0[0]3[1]. The lookup table size is 5 and the compression ratio is 47.91%. Overall, Technique 2 produces better results due to the increasing of occurrence frequencies of leftover characters while reducing the lookup table size. Figure 6(e) shows the best results based on Technique 3. In this case, 11[0]3[1] changes to 10[0]4[1], making the frequencies of blocks 10 and 4 increase. The final sequence is shorter than with the other techniques, the compression ratio is $CR = 72.92\%$, and the lookup table size is 4.

4. DECOMPRESSION

Huffman codes are prefix-free [Cormen et al. 2001]. This is an important property compared to other coding techniques used for compression and it significantly simplifies the decoding process. In Huffman code, the decoder easily recognizes the end of each codeword. An on-chip decoder at the serial input of the SoC under test is used to decompress test vectors. The decoder decodes input codes to recognize how many 1s or 0s need to be shifted into the scan

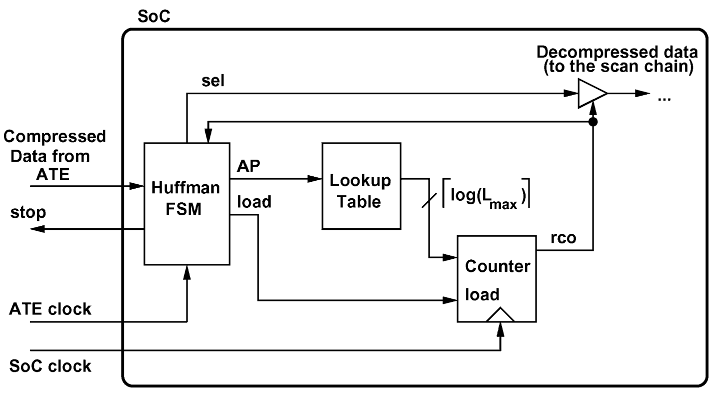


Fig. 7. RL-Huffman decoder architecture.

chain. The test vectors can be shifted into the scan chain with a higher rate, for example, at the SoC clock speed. The ATE sends the compressed data and clock for synchronization to the decoder. In our method the compressed data is a stream of variable-length (Huffman) codewords corresponding to the block lengths.

4.1 FSM-Based Decoder

Several Huffman decoders have been proposed in recent years which are independent of the circuit and data set [Chia-Hsing and Chein-Wei 1998; Benes et al. 1998; Rudberg and Wanhammar 1997]. The drawback, however, is that these decoders are expensive. To make decoder independent of the test set, the information of the Huffman tree can be transmitted with the compressed test data. However, such overhead has an adverse effect on the compression ratio. Moreover, such generic implementation is still expensive. In this article, we propose using an inexpensive decoder that is test set-dependent. Dependency of the decoder on the test data is not desirable in general but can be well justified by its low cost and generic architecture. Empirical results are shown in Section 6.

The block diagram of one possible implementation of a decoder is shown in Figure 7. The compressed data (input codeword) comes from the ATE to the Huffman FSM, where the input codewords are decoded. The decoded code addresses a small lookup table to find the block length L_i that indicates the number of 1s or 0s required to be shifted into the scan chain. The other outputs of the FSM are *sel* (to shift appropriate 0 or 1 into the scan chain), *load* (to enable of the counter), and *stop* (to stop the ATE when the counter has not finished its job while the FSM has generated a new address).

When the FSM generates an address pointer (AP), it enables *load* and the counter starts counting. At the same time *sel* ("0" or "1") is shifted into the scan chain through an open buffer. The FSM receives the new codeword for decoding. When the value of the counter and the output of the lookup table L_i become equal, the output of the counter (ripple carry-out (rco)) becomes 1 and the shifting *sel* value is stopped. It also signals the FSM to put the next address pointer on the input line of lookup table. If the FSM generates the address before receiving *rco* from the counter, it stops the ATE with *stop* signal and waits for *rco*. When it receives *rco* = 1 signal, it puts the new address on the output and deactivates the *stop* signal (*stop* = 0). Every time that a new AP is provided, a T flip-flop inside the FSM toggles to generate alternating blocks of 0s and 1s on the *sel* line.

4.2 CAM-Based Decoder

The decoder architecture of Figure 7 is one of many ways to implement the decoder. We would like to comment that other architectural styles are also efficient because the size of the decoder is quite small. In particular, content addressable memory (CAM) architecture can be used instead of the FSM-table pair to translate the codeword l_i to the block length L_i. Various efficient implementations of CAM have been proposed so far [McAuley and Francis 1993]. For large CAMs, the cost and power consumption of the large priority encoder inside the unit are matters of concern [McAuley and Francis 1993]. However, for applications such as ours that use small CAMs (up to a few hundred entries), these issues will not be problematic. In Section 6 we show the results of FSM-based implementation only. The CAM implementation of a few RL-Huffman decoders that we tried were only 7% to 12% more costly than their FSM-based counterparts [Tehranipour and Nourani 2002].

4.3 Test Time Reduction

Reducing the overall test application time is the ultimate goal of test pattern compression. In general, the amount of time reduction depends on the compression ratio and decompression method. In this section we estimate the overall time reduction ratio (TR).

Suppose the ATE and SoC under test work with frequencies f_{ATE} and f_{SoC}, respectively. When there is no compression, the test time is the same as the transfer time. That is:

$$T_{no\_comp} = \left(\sum_{i=1}^{N_b} L_i\right) \cdot \left(\frac{1}{f_{ATE}}\right).$$

When we compress the test data, such as in our method, the overall time is made of three portions:

$$T_{comp} = T_{transfer} + T_{decode} + T_{idle}.$$

In our method, the transfer and decode part is quite straightforward. Essentially, codewords (C_i) are transferred with a speed of f_{ATE} and counting toward L_i is done with a speed of f_{SoC}. In other words:

$$T_{transfer} = \left(\sum_{i=1}^{N_b} l_i\right) \cdot \left(\frac{1}{f_{ATE}}\right),$$

$$T_{decode} = \left(\sum_{i=1}^{N_b} L_i\right) \cdot \left(\frac{1}{f_{SoC}}\right).$$

Based on the decoder architecture shown in Figure 7, when the counter has not finished its counting for a block length L_i, the FSM cannot function on a new codeword C_i. The FSM receives l_i bit in $\frac{l_i}{f_{ATE}}$ cycles and the counter counts up to L_i in $\frac{L_i}{f_{SoC}}$ cycles. We may have $T_{idle} = 0$ only if the following inequality is satisfied:

$$\left(\max_{1\le i\le N_b}\{L_i\}\right)\cdot\left(\frac{1}{f_{SoC}}\right) \le \left(\min_{1\le i\le N_b}\{l_i\}\right)\cdot\left(\frac{1}{f_{ATE}}\right).$$

When this relation is not satisfied for the (l_i,L_i) pair, the idle time will be $\frac{L_i}{f_{SoC}} - \frac{l_i}{f_{ATE}}$ and therefore

$$T_{idle} = \sum_{i=1}^{N_b}\left(\frac{L_i}{f_{SoC}} - \frac{l_i}{f_{ATE}}\right), \quad \forall i \ni \frac{L_i}{f_{SoC}} > \frac{l_i}{f_{ATE}}.$$

Although the above relation can be approximated as $T_{idle} \simeq N_b\cdot(\frac{L_{avg}}{f_{SoC}} - \frac{l_{avg}}{f_{ATE}})$, we prefer to use the upper bound to combine T_{idle} with $T_{transfer}$ and T_{decode}. According to the above equation for T_{idle}, the upper bound will be $T_{idle} \le \sum_{i=1}^{N_b}\frac{L_i}{f_{SoC}}$ and therefore

$$T_{comp} \le \sum_{i=1}^{N_b}\left(\frac{l_i}{f_{ATE}} + 2\cdot\frac{L_i}{f_{SoC}}\right).$$

Finally, the time-reduction ration can be computed as

$$\begin{aligned} TR &= \frac{T_{no\_comp} - T_{comp}}{T_{no\_comp}} \\ &\ge \frac{\sum_{i=1}^{N_b}\frac{L_i}{f_{ATE}} - \sum_{i=1}^{N_b}\frac{l_i}{f_{ATE}} - 2\sum_{i=1}^{N_b}\frac{L_i}{f_{SoC}}}{\sum_{i=1}^{N_b}\frac{L_i}{f_{ATE}}} \\ &\ge \frac{\sum_{i=1}^{N_b}(L_i - l_i)}{\sum_{i=1}^{N_b}L_i} - 2\frac{\frac{1}{f_{SoC}}}{\frac{1}{f_{ATE}}} \\ &\ge CR - 2\frac{f_{ATE}}{f_{SoC}}. \end{aligned}$$

The above formula shows a lower bound for the *TR* metric. For the example, in Figure 1, if $f_{ATE} = 100$ MHz, $f_{SoC} = 1$ GHz, and $CR = 53.12\%$, we estimate $33.12\% \le TR \le 53.12\%$. In Section 6 we will show empirical evidence indicating that this lower bound of *TR* is not tight and, in practice, the relation $CR - 1.5\frac{f_{ATE}}{f_{SoC}} \le TR \le CR - 1.3\frac{f_{ATE}}{f_{SoC}}$ holds for large test sets.

5. POWER SAVING

The main goal of compression is to reduce the volume of test data. Each compression method decides on filling don't cares in the test set. Such a decision (sometimes called *test set mapping*) affects the power consumption of those components that finally receive uncompressed data from the decoder unit. In other words, the decoder simply performs the reverse job and creates the same data (including those that replaced don't cares) as decided in the compression process.

Reducing the number of transitions in test vectors reduces the switching activity and eventually power during the scan-in operation. Several techniques have been reported to reduce test application power. Test vector reordering [Girard et al. 1998] reorders scan cells such that the test sequence shifted in has the minimum switching activity during test application. The gated clock scheme [Bonhomme et al. 2001] reduces the clock rate on scan cells during shift operations to reduce scan power consumption. Scan latch partitioning in multiscan chains [Nicolici and Al-Hashemi 2000] reduces power dissipation by eliminating the spurious transitions which occur in the combinational part of the circuit via sending an extra test vector.

While scan power reduction is not the main focus of this article, our compression technique also reduces the *scan-in power*. This refers to the power consumption of elements in the scan chain during the scan-in operation. This desirable side effect (i.e., the reduced scan-in power) exists due to the inherent feature of our RL encoding technique that generates the minimum number of transitions. We have not pursued scan-out power (due to the core's output responses) in this article. Reducing scan-out power requires scan cell reordering [Dabholkar et al. 1998] or scan latch partitioning [Nicolici and Al-Hashemi 2000], which is beyond the scope of this article.

To analyze scan-in power, we have used the power estimation relation proposed in Sankaralingam et al. [2000] and Chandra and Chakrabarty [2002a]. These analytical relations only estimate bit transitions (proportional to power consumption) due to input test patterns traveling through the chain. Suppose n and m are the number of test patterns and scan chain length, respectively. Assume $T_i = (b_{i1}, b_{i2} \cdots b_{im})$ is the ith test pattern $(1 \leq i \leq n)$, where b_{i1} is the first bit scanned into the chain. Power dissipated in the scan cell elements (due to input test patterns) is estimated by counting the number of weighted transitions (WT) in the pattern, as presented in Sankaralingam et al. [2000]. The transition weight for a bit indicates how many times this bit is replaced with its complemented value when it is scanned into the scan chain.

In Chandra and Chakrabarty [2002a], the authors showed an analytical formula for scanning in pattern T_i, WT_i:

$$WT_i = \sum_{j=1}^{m-1} (m-j)\big(b_{ij} \oplus b_{i(j+1)}\big).$$

The total, average, and peak weighted transitions, due to input patterns, are

$$\begin{cases} WT = \sum_{i=1}^{n} WT_i, \\ WT_{avg} = WT/n, \\ WT_{peak} = MAX_{1 \leq i \leq n}\{WT_i\}. \end{cases}$$

As mentioned earlier, the RL encoding step in our technique builds minimum transitions in each test vector. The minimum bit transition for a test vector does not necessarily mean the minimum WT_i (as each bit enters the chain) in that

Table I. Comparing *WT* for an Example with Different Fillings of Don't Cares

i	$T_1 = 0xx11x0x$	Bit transition	WT
1	0xx11x01	≥2	≥ 8
2	00011000	2	8
3	00111000	2	9
4	00111100	2	8
5	01011x00	≥4	≥15
6	01111000	2	10
7	01111100	2	9
8	**00011100**	**2**	**7**

test vector. WT_i depends on the position of the transitions. Note that in the WT_i formula $m - j$ is the weight for the jth transition position in T_i. To reduce WT_i, we need to minimize $m - j$. If we push the transition position toward the least significant bit, the weighted transitions of a test vector (WT_i) and eventually the overall weighted transitions (WT) are reduced.

As an example, assume that $T_1 = (b_{11}, b_{12}, \ldots, b_{18}) = (0xx11x0x)$ when $m = 8$ and obviously the minimum number of transitions is two. Suppose that b_{11} is the first bit scanned into the scan chain. As shown in Section 2, after filling don't cares, RL encoding produces $T_1 = (00011100)$. This filling method gives the minimum weighted transition in each test vector. Fortunately, it is an inherent property of the RL encoding process to push minimum transitions toward the least significant bit. Any other different fillings with the same number of transitions cause larger WT. Table I shows the comparison between all possible fillings of don't cares while keeping the minimum number of transitions for test vector T_1. The last row in the table shows our filling process, which has the minimum WT. The other rows either show more WT or indicate more transitions, which eventually result in more WT. In Section 6 we compare average and peak scan-in power estimates with the random-fill technique.

6. EXPERIMENTAL RESULTS

The proposed technique is used to compress test data for ISCAS89 and PMC-Sierra's [PMC-Sierra 2002] benchmarks. Test patterns for PMC-Sierra's benchmarks are generated using FastScan [Mentor Graphics Corporation 2002] with static compaction. Test patterns for ISCAS89 benchmarks are identical to those used by other researchers [Chandra and Chakrabarty 2002b; Gonciari et al. 2002; Wolff and Papachristou 2002].

Figure 8 shows the results of our compression technique applied to 50 randomly generated test sets when we tuned the random filling to have 90%, 80%, and 70% don't cares. CR_{avg} shows the average compression ratios among 50 examples in each case and is close to D% especially when D% is high. This experiment confirms our conjecture that CR% is generally upper-bounded by D% (see Section 3.3).

Table II shows the compression results of five PMC-Sierra's SoCs and 12 ISCAS89 benchmarks assuming a $K = n \cdot m$ that implies unlimited block size. The total number of bits before (N_{bits_before}) and after (N_{bits_after}) compression

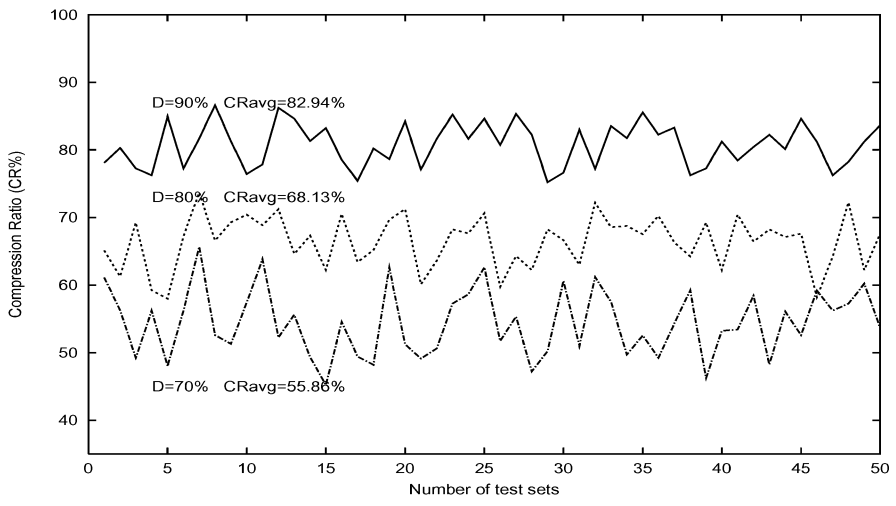


Fig. 8. CR for 50 random test sets.

Table II. Test Pattern Compression Analysis and Compression ($K = n \cdot m$)

Circuit	N_{bits_before}	$D\%$	N_{bits_after}	Compression ratio (CR %) RL-Huffman(+Tech.3)	FDR	MTC(+SLR)	VIHC
PMC1	164800	90	24209	85.31	—	—	—
PMC2	4557	88	712	84.36	—	—	—
PMC3	16830	83	3519	79.09	—	—	—
PMC4	89154	82	17295	80.60	—	—	—
PMC5	13311	81	2678	79.89	—	—	—
s420	1785	51	1026	42.52	—	—	—
s838	5762	59	2215	61.55	—	—	—
s1196	4448	56	3016	32.19	—	—	—
s1238	4864	56	3263	32.90	—	—	—
s1423	3276	46	2225	32.08	—	—	—
s5378	23754	71	10986	53.75 (**58.23**)	50.77	38.49 (46.01)	51.52
s9234	39273	72	20582	47.59 (52.46)	44.96	39.06 (47.20)	**54.84**
s13207	165200	92	28893	82.51 (**86.31**)	80.23	77.00 (81.07)	83.21
s15850	76986	83	25143	67.34 (**69.89**)	65.83	59.32 (64.59)	60.68
s38417	164736	68	59024	64.17 (**66.25**)	60.55	55.42 (58.56)	54.51
s38584	199104	82	74863	62.40 (**68.97**)	61.13	56.63 (63.41)	56.97
s35932	28208	35	3029	89.26 (92.07)	—	83.77 (**95.75**)	66.47

Note: FDR [Chandra and Chakrabarty 2002b]; MTC (+SLR) [Rosinger et al. 2001]; VIHC [Gonciari et al. 2002].

and don't care percentage ($D\%$) are also provided. This table also compares our results with the best results presented in Chandra and Chakrabarty [2002b], Rosinger et al. [2001], and Gonciari et al. [2002]. Specifically, for the last seven ISCAS89 benchmarks, we report our RL-Huffman method with and without using Technique 3. Recall that Technique 3 requires revisiting blocks to find higher-occurrence frequencies (equivalent to minimum number of blocks). The numbers in boldface indicate the highest compression ratio among those reported in the table. In most cases, Technique 3 improves the result of the original RL-Huffman by 2% to 7%.

Table III. Entropy Analysis

Circuit	RL-Huffman CR%	Entropy analysis		
		l_{avg}	H	E
s420	42.52	3.800	3.774	1.006
s838	61.55	4.343	4.321	1.005
s1196	32.19	3.081	3.011	1.023
s1238	32.90	3.160	3.098	1.020
s1423	32.08	3.387	3.319	1.020
s5378	53.75	3.599	3.556	1.012
s9234	47.59	4.223	4.180	1.010
s13207	82.51	5.328	5.299	1.005
s15850	67.34	4.466	4.429	1.008
s38417	64.17	4.121	4.104	1.004
s38584	62.40	4.271	4.249	1.005
s35932	89.26	3.908	3.873	1.009

Table IV. Test Time Analysis

Circuit	RL-Huffman CR%	Time reduction ratio ($TR\%$)		
		$\frac{f_{ATE}}{f_{SoC}} = \frac{1}{20}$	$\frac{f_{ATE}}{f_{SoC}} = \frac{1}{10}$	$\frac{f_{ATE}}{f_{SoC}} = \frac{1}{5}$
s420	42.52	35.62	28.72	14.92
s838	61.55	54.55	47.55	33.55
s1196	32.19	25.54	18.89	5.59
s1238	32.90	25.75	18.60	4.30
s1423	32.08	25.18	18.28	4.48
s5378	53.75	46.95	40.15	26.55
s9234	47.59	41.09	34.59	21.59
s13207	82.51	75.26	68.01	53.51
s15850	67.34	60.34	53.34	39.34
s38417	64.17	57.46	50.88	37.05
s38584	62.40	55.90	49.40	36.40
s35932	89.26	82.61	75.96	62.66

The FDR, MTC, and VIHC methods in Table II were chosen for comparison due to their similarities in using run-length encoding. We acknowledge that other compression methods, such as the dictionary-based LZ77 [Wolff and Papachristou 2002] or LZW [Knieser et al. 2003] approaches, have reported higher compression ratios for some of these benchmarks. However, their results in terms of consistency to achieve a high compression ratio, cost of memory-demanding decoder, and scan-in power due to high bit transitions are still inconclusive.

Table III shows the entropy analysis of ISCAS89 benchmarks. As shown in the last three columns, using our technique l_{avg}, H, and E are very close to their corresponding lower bound.

Table IV shows the test time analysis of ISCAS89 benchmarks for different frequencies of the system. It shows that, when $f_{SoC} \gg f_{ATE}$, the test application time reduction ratio (TR) is close to CR, confirming our analysis in Section 4.3.

As Figure 7 shows, the decoder consists of mainly three components: one FSM, a lookup table, and a counter. The counter size is $\lceil log_2 L_{max} \rceil$ bit, where

Table V. Decoder (FSM-Table) Cost Analysis

Circuit	$L_{\max}$	Cost [NAND]
PMC2	217	481
PMC3	165	373
s420	47	173
s1196	32	161
s1423	214	432
s5378	281	551
s9234	296	589
s15850	611	769

Table VI. Tradeoff Between Compression Ratio and Decoder Cost Using Different Maximum Block Size (K)

			Maximum run length (K)					
Circuit	Techniques	Factors	4	8	12	16	32	$K = n \cdot m$
PMC2	1	*RL-Huffman CR%*	52.5	65.4	68.0	75.7	78.2	84.3
		FSM-table Cost	48	91	147	251	379	481
	2	*RL-Huffman CR%*	59.0	76.5	**80.2**	79.6	79.1	**84.3**
		FSM-table Cost	51	70	**82**	121	239	**481**
s5378	1	*RL-Huffman CR%*	31.2	39.5	45.9	49.2	51.1	53.7
		FSM-table Cost	43	93	167	301	412	551
	2	*RL-Huffman CR%*	37.1	41.8	47.1	50.2	52.4	53.7
		FSM-table Cost	49	89	148	277	385	551
s9234	1	*RL-Huffman CR%*	22.8	29.1	34.7	42.7	42.4	47.6
		FSM-table Cost	44	101	158	337	431	589
	2	*RL-Huffman CR%*	28.5	34.2	38.9	40.6	43.2	47.6
		FSM-table Cost	51	103	155	301	417	589a

$L_{max} = \max_{1 \le i \le N_l}\{L_i\}$. The counter size/cost is not sensitive to compression factors. Therefore, we report only the cost of the main part of the decoder, that is, the FSM with the lookup table. This cost (FSM-table pair) is summarized in Table V for some of ISCAS89 and PMC-Sierra's benchmarks based on the number of equivalent *NAND* gates reported by the Synopsys synthesizer [Synopsys, Inc. 2002].

Table VI shows the effect of maximum run-length K on the compression ratio and the cost of FSM-table pair. To limit the maximum run-length we explored two techniques (Techniques 1 and 2) discussed in Section 3. As shown, the second technique achieves an overall higher compression rate and smaller decoder size than the first technique. This is because the second technique decomposes blocks intelligently to increase the occurrence frequency of characters. The compression ratio in the case of $K = n \cdot m$ for each circuit is more than the case when K is limited. When $K = n \cdot m$, the longer block lengths will be sent by much smaller codewords. This results in more compression, but the cost of the decoder slightly increases.

Table VI clearly shows the tradeoff possibility between CR and the FSM-table cost for Techniques 1 and 2 by choosing various K's. By choosing a smaller K, we significantly reduce the cost of a decoder with a slight sacrifice of compression

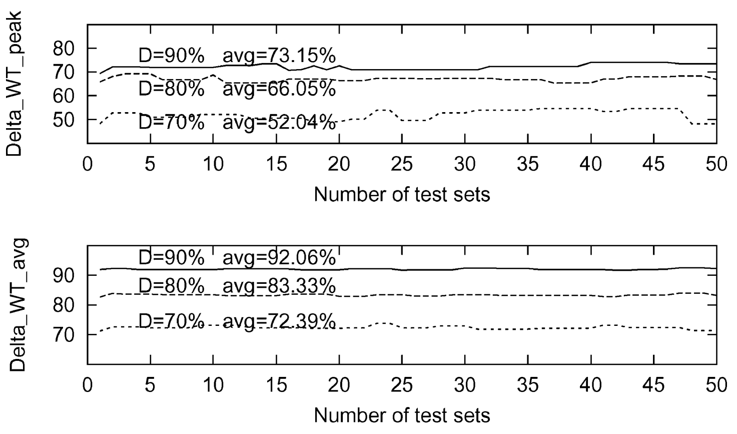


Fig. 9. ΔWT_{peak} and ΔWT_{avg} for 50 random test sets.

Table VII. Comparing Switching Activities (Estimate of Scan-In Power Dissipation)

Circuit	$N_{patterns}$	Randomly Filled		RL-Huffman		Scan-in Power Reduction	
		WT_{avg}	WT_{peak}	WT_{avg}	WT_{peak}	$\Delta WT_{avg}\%$	$\Delta WT_{peak}\%$
s420	1785	200	314	95	238	52.36	24.20
s838	5762	700	1310	219	1022	68.72	21.98
s1196	4448	244	393	52	234	78.69	40.45
s1238	4864	241	356	49	177	79.47	50.28
s1423	3276	1860	2551	831	2098	55.28	17.75
s5378	23754	11409	13370	3433	11519	69.90	13.84
s9234	39273	15085	18416	3957	14092	73.76	23.47
s13207	165200	122034	137455	7734	94879	93.66	30.97
s15850	76986	92212	102816	13513	70875	85.34	31.06
s38417	164736	581505	65560	116301	411718	80.51	37.26
s38584	199104	527871	572499	85655	481158	83.77	15.95
s35932	28208	316548	787718	39874	107226	87.40	86.38

ratio. For example, consider the statistics in boldface obtained for the PMC2 benchmark. We can trade off the 4.1% compression ratio by reducing the size of decoder by a factor of 6.

Figure 9 shows the reduction of average (ΔWT_{avg}) and peak (ΔWT_{peak}) switching activities for various random test sets. Statistics are reported based on the results of 50 randomly generated test sets for average and peak values. Each test set includes 100 test vectors and each vector is 32 bit for different don't care percentages (70%, 80%, and 90%). When D% increases, our method is more efficient in reducing the average and peak power.

Table VII shows the comparison between scan-in power dissipated (due to the input test patterns) in two cases. First, we replace the don't care bits randomly for each ISCAS89 benchmark. This process is performed 50 times and results shown in the table are the average of the 50 times compilations. Second, the don't cares are replaced according to our technique. The average and peak switching activities are shown in the table. The reduction percentage for weighted transition average and peak (ΔWT_{avg} and ΔWT_{peak}) are shown in the last two columns with respect to random filling. The average and peak power reductions reported in Table VII are very close to those reported in Chandra and Chakrabarty [2002b] and Rosinger et al. [2001]. This is expected, as in all three methods the run-length encoding produces the minimum bit transition.

7. CONCLUSION

We presented a new compression and decompression technique based on run-length and Huffman codings for scan testing to reduce test data volume, test application time, and scan-in power. The proposed technique takes advantage of don't cares generated by the ATPG. The method can be tuned by limiting the maximum run-length to tradeoff compression ratio and decoder cost. Experimental results show that up to a 89% compression ratio and 93% scan-in power reduction is achievable for the benchmarks that we have tried so far.

ACKNOWLEDGMENTS

The authors wish to thank Karim Arabi (PMC Sierra), Christos Papachristou and Francis Wolff (Case Western Reserve University) and Michael Knieser (Purdue University Indianapolis) for generously sharing their test pattern sets with us. We also thank Krishnendu Chakrabarty (Duke University) for his fruitful discussion on scan power analysis.

REFERENCES

BAYRAKTAROGLU, I. AND ORAILOGLU, A. 2001. Test volume and application time reduction through scan chain concealment. In *Proceedings of the Design Automation Conference* (DAC'01). 151–155.

BENES, R., NOWICK, S., AND WOLF, A. 1998. A fast asynchronous Huffman decoder for compressed-code embedded processors. In *proceedings of the Fourth International Symposium on Advanced Research in Asynchronous Circuits and Systems*, 43–56.

BONHOMME, Y., GIRARD, P., GUILLER, L., LANDRAULT, C., AND PRAVOSSOUDOVTCH, S. 2001. A gated clock scheme for low power scan testing of logic ICs or embedded cores. In *Proceedings of the VLSI Test Symposium* (VTS'01). 253–258.

CHANDRA, A. AND CHAKRABARTY, K. 2000. Test data compression and decompression for system-on-a-chip using Golomb codes. In *Proceedings of the VLSI Test Symposium* (VTS'00). 113–120.

CHANDRA, A. AND CHAKRABARTY, K. 2001. Frequency-directed run-length (FDR) codes with applications to system-on-a-chip test data compression. In *Proceedings of the VLSI Test Symposium* (VTS'01). 42–47.

CHANDRA, A. AND CHAKRABARTY, K. 2002a. Low-power scan testing and test data compression for system-on-a-chip. *IEEE Trans. Comput.-Aided Des.*, *21*, 5, 597–604.

CHANDRA, A. AND CHAKRABARTY, K. 2002b. Reduction of SOC test data volume, scan power and testing time using alternating run-length codes. In *Proceedings of the Design Automation Conference* (DAC'02). 673–678.

CHIA-HSING, L. AND CHEIN-WEI, J. 1998. Low power parallel Huffman decoding. *Electron. Lett. 34*, 3, 240–241.

CORMEN, T., LEISERSON, C., RIVEST, R., AND STEIN, C. 2001. *Introduction to Algorithms*. McGraw-Hill, New York, NY.

COVER, T. AND THOMAS, J. 1991. *Elements of Information Theory*, Wiley, New York, NY.

DAS, D. AND TOUBA, N. 2000. Reducing test data volume using external/LBIST hybrid test patterns. In *Proceedings of the International Test Conference* (ITC'00). 115–122.

DABHOLKAR, V., CAKRAVARTY, S., POMERANZ, I., AND REDDY, S. 1998. Techniques for minimizing power dissipation in scan and combinational circuits during test application. *IEEE Trans. Comput.-Aided Des. 17*, 12, 1325–1333.

Girard, P., Landrault, C., Pravossoudovtch, S., and Severac, D. 1998. Reducing power consumption during test application by test vector ordering. In *Proceedings of the International Symposium on Circuits and Systems* (ISCAS'98). vol. 2. 296–299.

Golomb, S. 1966. Run-length encoding. *IEEE Trans. Inform. Theory IT-12*, 399–401.

Gonciari, P., Al-Hashimi, B., and Nicolici, N. 2002. Improving compression ratio, area overhead, and test application time for system-on-a-chip test data compression/decompression. *Proceedings of the Design, Automation and Test in Europe* (DATE'02). 604–611.

Hamzaoglu, I. and Patel, J. 2000. Reducing test application time for built-in self-test patterns. In *Proceedings of the VLSI Test Symposium* (VTS'00). 369–376.

Hellebrand, S., Rajski, J., Tarnick, S., Venkataraman, S., and Courtois, B. 1995. Built-in test for circuit with scan based on reseeding of multiple-polynomial linear feedback shift registers. *IEEE Trans. Comput. 44*, 223–233.

Hsu, F., Butler, K., and Patel, J. 2001. A case study on the implementation of the illinois scan architecture. In *Proceedings of the International Test Conference* (ITC'01). 538–547.

Huffman, D. 1952. A method for the construction of minimum redundancy codes. In *Proc. IRE*. *40*, 9, 1098–1101.

Iyengar, V., Chakrabarty, K., and Murray, B. 1998. Built-in self testing of sequential circuits using precomputed test sets. In *Proceedings of the VLSI Test Symposium* (VTS'98). 418–423.

Jas, A., Ghosh-Dastidar, J., and Touba, N. 1999. Scan vector compression/decompression using statistical coding. In *Proceedings of the VLSI Test Symposium* (VTS'99). 114–120.

Khoche, A., Volkerink, E., Rivoir, J., and Mitra, S. 2002. Test vector compression using EDA-ATE synergies. In *Proceedings of the VLSI Test Symposium* (VTS'02). 97–102.

Knieser, M., Wolff, F., Papachristou, C., Weyer, D., and McIntyre, D. 2003. A technique for high ratio LZW compression. In *Proceedings of the Design, Automation and Test in Europe* (DATE'03). 116–121.

Koenmann, B. 2000. "SMARTBIST," Presentation by IBM in ITC.

Krishna, C., Jas, A., and Touba, N. 2001. Test vector encoding using partial LFSR reseeding. In *Proceedings of the International Test Conference* (ITC'01). 885–893.

McAuley, A. and Francis, P. 1993. Fast routing table lookup using CAMs. In *Proceedings of the INFOCOM'93*.

Mentor Graphics Corporation 2002. *User manuals for fastScan*. Mentor Graphics Corporation.

Nicolici, N. and Al-Hashemi, B. 2000. Scan latch partitioning into multiple scan chains for power minimization in full scan sequential circits. In *Design, Automation and Test in Europe* (DATE'00). 715–722.

PMC-Sierra 2002. www.PMC-Sierra.com.

Rabaey, J. 2002. *Digital Integrated Circuits*, Prentice-Hall, Englewood Cliffs, NJ.

Rajski, J. et. al. 2002. Embedded deterministic test for low cost manufacturing test. In *Proceedings of the International Test Conference* (ITC'02). 301–310.

Reddy, S., Miyase, K., Kajihara, S., and Pomeranz, I. 2002. On test data volume reduction for multiple scan chain designs. In *Proceedings of the VLSI Test Symposium* (VTS'02). 103–108.

Rosinger, P., Gonciari, P., Al-Hashimi, B., and Nicolici, N. 2001. Simultaneous reduction in volume of test data power dissipation for systems-on-a-chip. *Electron. Lett.*, *37*, 24, 1434–1436.

Rosinger, P., Gonciari, P., Al-Hashimi, B., and Nicolici, N. 2002. Analysing trade-offs in scan power and test data compression for system-on-a-chip. *IEE Proc. Comput. Dig. Techniq. 149*, 4 (July), 188–196.

Rudberg, M. and Wanhammar, L. 1997. High speed pipelined parallel Huffman decoding. In *Proceedings of the International Symposium on Circuits and Systems* (ISCAS'97), vol. 3. 2080–2083.

Sankaralingam, R., Oruganti, R., and Touba, N. A. 2000. Static compaction techniques to control scan vector power dissipation. In *Proceedings of the VLSI Test Symposium* (VTS'00). 35–40.

Synopsys, Inc. 2002. *User Manuals for SYNOPSYS Toolset, Version.05*. Synopsys, Inc.

Tehranipour, M. H., Ahmed, N., and Nourani, M. 2003. Testing SoC interconnects for signal integrity using boundary scan. In *Proceedings of the VLSI Test Symposium* (VTS'03). 4A.13–4A.18.

Tehranipour, M. H. and Nourani, M. 2002. Compression techniques for scan. Tech. rep. UTD-10-05-2002, Dept. of EE, University of Texas at Dallas, Richardson, TX.

Wang, S. and Chiou, S. 2001. Generating efficient tests for continuous scan. In *Proceedings of the Design Automation Conference* (DAC'01). 162–165.

Wang, S. and Gupta, S. 1997. ATPG for heat dissipation minimization during scan testing. In *Proceedings of the Design Automation Conference* (DAC'97). 614–619.

Wolff, F. and Papachristou, C. 2002. Multiscan-based test compression and hardware decompression using LZ77. In *Proceedings of the International Test Conference* (ITC'02). 331–339.

Wurtenberger, A., Tautermann, C., and Hellebrand, S. 2003. A hybrid coding strategy for optimized test data compression. In *Proceedings of the International Test Conference* (ITC'03). 451–459.

Zorian, Y., Marinissen, E., and Dey, S. 1999. Testing embedded-core-based system chips. *Computer, 32*, 6, 52–60.

Received March 2003; revised July 2003, January 2004; accepted January 2004

A 4-Geometry Maze Router and Its Application on Multiterminal Nets

GENE EU JAN
National Taipei University
KI-YIN CHANG
National Taiwan Ocean University
and
SU GAO and IAN PARBERRY
University of North Texas

The maze routing problem is to find an optimal path between a given pair of cells on a grid plane. Lee's algorithm and its variants, probably the most widely used maze routing method, fails to work in the 4-geometry of the grid plane. Our algorithm solves this problem by using a suitable data structure for uniform wave propagation in the 4-geometry, 8-geometry, etc. The algorithm guarantees finding an optimal path if it exists and has linear time and space complexities. Next, to solve the obstacle-avoiding rectilinear and 4-geometry Steiner tree problems, a heuristic algorithm is presented. The algorithm utilizes a cost accumulation scheme based on the maze router to determine the Torricelli vertices (points) for improving the quality of multiterminal nets. Our experimental results show that the algorithm works well in practice. Furthermore, using the 4-geometry router, path lengths can be significantly reduced up to 12% compared to those in the rectilinear router.

Categories and Subject Descriptors: B.7.2 [**Integrated Circuits**]: Design Aids—*Placement and routing*

General Terms: Algorithms, Design, Performance

Additional Key Words and Phrases: Maze router, λ-geometry, cell map, Steiner tree

This work was supported by the National Science Council of Taiwan, ROC, under grant NSC-87-2211-E-019-004.

Authors' addresses: G. E. Jan, Department of Computer Science, National Taipei University, Sun Shia, Taipei County, 237, Taiwan; email: gejan@mail.ntpu.edu.tw; K.-Y. Chang, Department of Merchant Marine, National Taiwan Ocean University, Keelung, 202, Taiwan; email: b0170@mail.ntou.edu.tw; S. Gao, Department of Mathematics, University of North Texas, P. O. Box 311430, Denton, TX 76203; email: sgao@unt.edu; I. Parberry, Department of Computer Science, University of North Texas, P. O. Box 13886, Denton, TX 76203; email: ian@cs.unt.edu.

1. INTRODUCTION

The original objective of maze routing problems is to find a shortest path between two terminal cells on a rectangular grid of cells without crossing any obstacles. The first maze routing algorithm was presented by Lee [1961]. To this date, Lee's algorithm and its variations are still widely used in VLSI and PCB designs, in maze games, and in road map routing problems [Rubin 1974; Hoel 1976; Lin et al. 1990; Fawcett and Robinson 2000]. The popularity of Lee's algorithm lies in its simplicity and the guarantee it offers of finding a shortest path if one exists. However, Lee's algorithm is intrinsically based on the 2-*geometry* (also known as the *Manhattan geometry, rectilinear geometry,* etc.) of the grid plane. Each cell is considered to have only four neighbors, corresponding to at most four directions (left, right, up, and down) to move along an admissible path. This restriction is natural for some of the applications of the algorithm (e.g. VLSI designs), but unnecessary for most of the other situations.

To get around this restriction, many researchers took a graph-based approach to the problems arising in fields such as robot motion planning [Dí-az de León S. and Sossa A. 1998; Barraquand and Latombe 1991], computer-aided design [Xing and Kao 2002], and pattern recognition [Kimmel et al. 1995; Ogniewicz and Kubler 1995]. These graph-based approaches with suitable search structures solve the problems in substantially different ways. Some efficient implementations of Dijkstra's single-source shortest-path algorithm [Dijkstra 1959] have been proposed using Fibonacci heaps (F-heaps) to find the shortest path in the graph model [Ahuja et al. 1990; Henzinger et al. 1997]. For a grid with N cells, the implementation of Fibonacci heaps uses up to N F-heaps and the total complexity, including preprocessing, has not been improved to $O(N)$. Some other complicated data structures such as priority queues, hierarchical buckets, or multilevel buckets are required to avoid the sorting bottleneck even when the edge weights are bounded in most of these articles [Dinic 1978; Goldberg 2001; Thorup 1999]. In a computer-game application, the situation is naturally presented in a raster plane and the grid-based approach is easier to implement. Also, two advantages are obtained: independence from obstacles in the search process and freedom of preprocessing to construct a suitable search structure. In this article, we present a maze routing algorithm on the grid plane based on the 4-geometry. Our proposed algorithm provides simple and easy implementation when compared with other graph models, since it takes advantages of the nature of grid planes or raster planes and limited weights in λ-geometry (1 only in 2-geometry; 1 and $\sqrt{2}$ only in 4-geometry, 1; $\sqrt{2}$, $\sqrt{5}$ only in 8-geometry, and so on so forth). The algorithm is a directional improvement of Lee's algorithm, and therefore enjoys the two main advantages of Lee's algorithm: it is obstacle-independent and it guarantees finding the shortest path if one exists. Lee's algorithm fails for 4-geometry if different distances occur. An example will be given in Section 2 below. A straightforward attempt to improve Lee's algorithm by using equal cost wavefronts causes a substantial increase in time complexity [Fawcett and Robinson 2000]. Our algorithm works in a general context and uses a different data structure than that of Lee's algorithm. But in the case of 2-geometry, the two algorithms are the same. Also, our algorithm has the same time and space complexities of $O(N)$ as Lee's algorithm, where N is the number of cells in the grid plane. In fact, our algorithm uses at most twice the time as Lee's algorithm. Empirical data has shown that the overhead in running time is easily absorbed by the enhanced computing capabilities of modern computers.

It is worth mentioning that although we focus on the 4-geometry in this article because VLSI designers have no concern for the geometry value that is greater than 4, the algorithm works in more general situations. It can be easily adapted to handle general λ-geometry for $\lambda > 4$, and it can be used without substantial modification for higher dimensions. Thus considering 4-geometry is not an intrinsic restriction for us, although considering 2-geometry is an intrinsic restriction to applying Lee's algorithm.

Once the 4-geometry maze router has been introduced, the heuristic algorithm based on the maze router for the obstacle-avoiding Steiner tree problem will be presented. The main objective of this part of the article will be to implement both the 2-geometry and the 4-geometry maze routers in the multiterminal nets. The problem of optimally routing a multiterminal net in the presence of obstacles has received less attention. As a result, VLSI designers typically use a multiterminal variant of a maze routing algorithm and usually produce solutions that are far from optimal [Ganley and Cohoon, 1994]. Thus, one of the objectives of this study is to improve the quality of the multiterminal nets in the maze router. Ganley and Cohoon [1994] first presented an efficient model that allows computation of optimal obstacle-avoiding rectilinear (2-geometry) Steiner trees with three or four terminals. For nets with five or more terminals, they presented heuristic algorithms that work well in both theory and practice. Also, several researchers have successfully worked on the 1-Steiner tree problem [Georgakopolulos and Papadimitrious, 1987; Cieslik, 1991] and the heuristic 1-SMT (Steiner Minimal Trees) problem with a minimum spanning tree [Kahng and Robins, 1992; Cieslik, 1998]. Based on the concept of the 1-Steiner vertex (point) approach and the characteristics of the maze router, this article introduces a new cost accumulation scheme in the grid plane to obtain the Torricelli vertex, the minimum sum of the costs from all the terminals, for improving the quality of the multiterminal nets in global and local routings. The experimental data shows that the results are near optimal. Furthermore, experimental data has also shown that, by using the 4-geometry, path lengths can be reduced around 10–12% compared to those in the 2-geometry. This is a great improvement in many realistic applications by using the maze router. Finally, the need to design high-performance circuits has also called for the consideration of non-2-geometries (also known as *nonrectilinear* geometries) [Sherwani 1999; Teig 2002] in the field of VLSI design. This shows that there is a high potential in the applications of our 4-geometry router. An interesting observation is that the length reduction is quite marginal for higher geometries. Therefore it is effective enough to apply the 4-geometry for global routing problems. But the higher geometry is very useful for the other fields where more selective directions are needed to emulate the real routing situation.

The article is organized as follows. In Section 2 we give a description of our problem and formulate it in rigorous terms. In Section 3 the data structure is first described and defined. Then, the 4-geometry maze router is presented and the correctness and performance of the algorithm is proved and analyzed. The modification of the algorithm for the higher-geometry maze router is described as well. Some experimental results for the 4-geometry and 8-geometry routers are demonstrated. In Section 4 a heuristic algorithm based on the maze router is proposed for the multiterminal nets. Starting from a Torricelli vertex, the algorithm will search and connect a near optimal Steiner tree. After that, a cost accumulation scheme for subsets of three or four terminals among all terminals is included to refine the quality of the routing result. The experimental results show good solutions for obstacle-avoiding multi-terminal nets. The conclusions are summarized in Section 5.

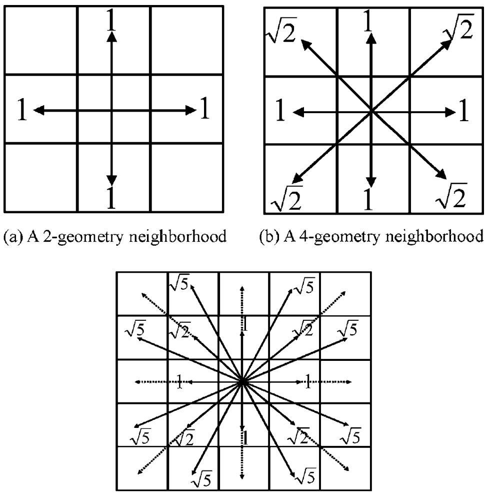
2. PROBLEM FORMULATION AND DESCRIPTION

Given an $m \times n$ rectangular grid E of N cells, a *cell map* is a function with domain E and a finite set of values that will be described in Section 3.1. The cell map indicates which cells constitute obstacles at the very least. As the algorithm unfolds, the cell map can also be used to indicate the intermediate status of the cells. A path on E is simply a sequence of cells on E. A path $p = (c_1, c_2, \dots, c_n)$ is *admissible* if for each $i = 1, \dots, n-1, c_i$ and c_{i+1} are neighboring cells and neither of them is an obstacle cell. It should also be given a monotone path cost function f, defined on the set of all paths in E. According to Lee, a function is *monotone* if for any path p and subpath q, $f(q) \leq f(p)$. For a given source cell S and a destination cell D, the *single-source routing problem* involves finding an admissible path p from S to D with the smallest possible value $f(p)$.

Note that the formulation of our problem is not in its most general form. This is done for the convenience of the reader and for simplicity in our presentation. In fact, the path cost function can in general be a vector function $F = (f_1, \dots, f_r)$ with its values ordered lexicographically. Also our algorithm will generalize to three-dimensional grids without any difficulty.

In his original article [Lee 1961], Lee claimed to have solved the routing problem completely. But in fact Lee's algorithm does not work for the most general case [Rubin 1974]. In the rest of this section we give an example of a cell map for which Lee's algorithm does not yield the correct answer.

For an integer $\lambda \geq 2$, the λ-geometry allows edges with the angles of $i\pi/\lambda$, for all i, $\lambda = 2, 4, 8, 16$ and ∞ corresponding to rectilinear (90°), 45°, 22.5°, 11.25°, and Euclidean geometries, respectively. For a 2-geometry neighborhood, depicted in Figure 1(a), we are only interested in the cells above, below, to the left, and to the right of a cell, that is, all of the cells that are distance 1 unit from the center cell. The neighboring cells thus share an edge. If the four cells on the diagonal are included, we are working in a 4-geometry neighborhood, as depicted in Figure 1(b). In the 4-geometry, each cell has eight neighbors, and thus movements along the diagonal directions are possible. Note that Lee's algorithm can also be applied to the situation where each cell has eight neighbors. But, in order for Lee's algorithm to work, all eight neighboring cells must be equidistant from the center cell. In the 8-geometry neighborhood, each cell has 24 related neighbors. The distance of the 24-cell connected neighborhoods to the



(a) A 2-geometry neighborhood (b) A 4-geometry neighborhood

(c) An 8-geometry neighborhood

Fig. 1. Cell connection styles.

center cell is shown in Figure 1(c). There are 16 solid-line reachable neighbors that must be computed for each move. However, the eight dashed-line reachable neighbors do not need to be calculated since they can be extended by the next computation and will be computed in the next move. Note that each angle in the 8-geometry is not exactly 22.5°, half of 45°, because the angle is divided in the grid plane, not in a circular plane.

For the 16-geometry neighborhood, the running time for computation and condition statements is about twice that for the 8-geometry. However, the 16-geometry routing has twice the selective directions for searching a shortest path than that of the 8-geometry.

Lee's algorithm can be visualized as a wave propagation process. Putting the source cell in list L initializes the search. Two lists L and L_1 are defined to keep track of the cells on the wavefront (also called *frontier cells*) and their neighboring cells, respectively. After all the neighboring cells in L are included in L_1, the list L_1 is processed so that an expanded wavefront is found. Then any cell in L is deleted if all of its neighboring cells have been processed (updated) and L is updated by this new wavefront. The search is terminated if the destination cell is reached. This causes incorrect searching results if any single step is not equidistant. A simple example with a polygonal obstacle is shown in Figure 2. The result shows that the path p reaches the destination D first with the distance of 128.1 if Lee's algorithm is applied to the *4-geometry*. The path p' only reaches point A when path p reaches destination D. In fact, the shortest path from S to D is an extension of the path p' and has a shorter distance of 115. Thus Lee's algorithm no longer works for the 4-geometry.

In previous work, the radiation scheme was used for the conventional maze routing with diagonal connections to obtain the distances from the eight directional neighboring cells. The algorithm wasn't terminated until the AT values of the entire cells were the same in two consecutive steps if the obstacles of

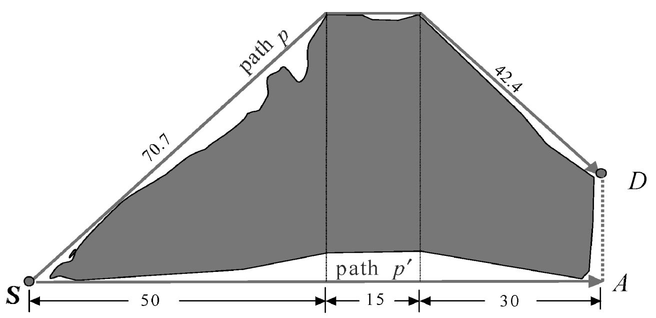


Fig. 2. Implementation of Lee's algorithm in the 4-geometry.

U-shape existed. Thus, the worse-case complexity was $O(N^2)$ [Jan et al. 1997]. Furthermore, a variant of Lee's router uses a time-ordered (cost-ordered) list to store the wavefront, rather than a simple list L. But the time complexity of their algorithm was described as polynomial in the number of edges [Fawcett and Robinson 2000]. In the next section, we solve the problem by introducing the concept of buckets. This data structure enables us to perform uniform propagation for arbitrary cost functions.

3. REQUIRED DATA STRUCTURE AND THE MAZE ROUTING ALGORITHM

The required data structure for this algorithm is mainly a cell map, and some linked lists and buckets. In the cell map, the cost function is represented by a so-called *AT*, or time of arrival, value. This follows a standard practice since the algorithm is first implemented for searching shortest path on the raster electronic map. The data structure and detail of the 4-geometry maze routing algorithm are introduced since it is of concern to most VLSI designers. The 8-geometry, 16-geometry, etc, have received less attention, but are beneficial for the other areas of application that will be discussed in the last subsection.

3.1 The Cell Map

The number of data fields in the cell map is flexible and depends on the requirements of individual applications. There are at least three parameters for cell storage in the 4-geometry, that is, F/O, *AT*, and *Vis*. The F/O (*Free space or Obstacle*) parameter distinguishes whether a cell is an obstacle, in which case the value is infinity, or a passable area, in which case the value is 1. The *AT* parameter stores the time (or other types of cost; in this article the *AT* value equals the distance divided by the velocity) needed to travel from the source cell to the current cell, and its initial value is infinity. The third parameter, *Vis* (*Visited*), distinguishes whether the cell has visited all its neighbors and update their *AT* values, and its initial Boolean value is *false*. For the initial cell condition, the black cells represent obstacles and the white cells represent passable areas. If the F/O parameter of any area has a finite value between 1 and infinity, we are working on the optimal path of various terrains. In an $m \times n$ grid plane, any cell $C_{i,j}$ has three parameters, $F/O_{i,j}$, $AT_{i,j}$, and $Vis_{i,j}$, where $0 \leq i \leq m-1, 0 \leq j \leq n-1$.

3.2 The Algorithm for 4-Geometry Maze Router

For an $m \times n$ grid of cells, the first step in the algorithm is to input a source cell (expressed by S) and a destination cell (expressed by D). According to the algorithm, the AT value of the source cell is zero and the indices of the source cell S will be moved into the first bucket, LL_0. The remaining cells that spread from S will be moved into their corresponding buckets according to their AT values. The AT value of any cell would be increased at most by $\sqrt{2}$ from its neighboring cell. The number of decimal digits in the irrational number, $\sqrt{2}$, is determined by the total number of cells on the grid plane. Therefore, the number of buckets can be reduced to three (LL_0, LL_1, LL_2) for the purpose of recycling since the updated cells would be moved into one of the next two buckets. In addition, a temporary list TL is applied to the algorithm to store the indices (i, j) of visited cells until all of the cells in the current bucket have been processed.

Since we have replaced the buckets β with three circular buckets, the three buckets are marked as LL_{index}, where the *index* is an integer variable and $0 \leq index \leq 2$. We also define a function Update_*AT* to support the structured program. The purpose of the function Update_*AT* is to update the $AT_{i,j}$ value of $C_{i,j}$. There are two input parameters in this function named (i, j) and $new\_AT_{i,j}$, the updated $AT_{i,j}$.

```
Function Update_AT ((i, j), new_AT_{i,j}):
Step 1: Update its $AT_{i,j}$ value of $C_{i,j}$ for an $m \times n$ grid plane.
If $new\_AT_{i,j}$ is less than $AT_{i,j}$, then $AT_{i,j} = new\_AT_{i,j}$.
END {Function of Update_AT}
```

The 4-geometry routing algorithm:

```
Initialization:
        For each cell $C_{i,j}(F/O_{i,j}, AT_{i,j}, Vis_{i,j})$ in an $m \times n$ grid plane, the initial $F/O_{i,j}$
        value is 1 if cell $C_{i,j}$ is in the free space or $\infty$ if it is in the obstacle. $AT_{i,j} = \infty$
        and $Vis_{i,j}$ = false for all cells, where $0 \leq i \leq m-1, 0 \leq j \leq n-1$. The initial
        value of index is 0.
Step 1: Input the coordinates of the source cell $S$ and the destination cell $D$. If $F/O_{i,j}$
        of the source cell is not equal to $\infty$, then update $AT_{i,j} = 0$, else return the
        error message "The source cell is in the obstacle" and terminate. If $F/O_{i,j}$ of the
        destination cell is equal to $\infty$, then return the error message "The destination
        cell is in the obstacle" and terminate.
Step 2: Compute the time of arrival between the source cell and the remaining cells.
        Step 2.1: Move the source cell $S$ into the temporary list $TL$.
        Step 2.2: Move the source cell $S$ into the bucket $LL_0$.
        Step 2.3: For each cell in the $LL_{index}$, update the $AT$ values of its neighboring
                  cells. Meanwhile, if the destination cell is in the current bucket, then
                  break step 2.
                  Step 2.3.1: Remove the indices of the first cell $C_{i,j}$ from the front end
                              of the $LL_{index}$ and update this cell's $Vis_{i,j}$ to true.
                  Step 2.3.2: Update the time of arrival ($AT_{i,j}$) of the 4-geometry neigh-
                              bors of cell $C_{i,j}$.
                              For each $C_{i,j}$'s neighbor $C_{i',j'}$, if the neighbor's $F/O = 1$
                              then call Update_AT ((i', j'), $AT_{i,j} + 1$) for the 2-geometry
                              neighbors or call Update_AT ((i', j'), $AT_{i,j} + \sqrt{2}$) for the
                              four diagonal neighbors.
```

```
            Step 2.3.3: Iterations
                        If $LL_{index}$ is not empty, then go back to step 2.3.
    Step 2.4: Move the cells' indices in the $TL$ into their corresponding buckets.
            Step 2.4.1: For all the indices in the $TL$, move $(i, j)$ from $TL$ into
                        $LL_{\lfloor AT_{i,j} \rfloor \bmod 3}$.
            Step 2.4.2: If the $TL$ is empty, then update the $index$ value,
                        $index = (index+1) \bmod 3$.
    Step 2.5: Iterations
              If two consecutive buckets are not empty, then go back to step 2.3.
Step 3: Backtracking
        If the $AT_{i,j}$ value of the destination cell is infinity, return the error message:
        "There is not any path between this given pair". Otherwise backtrack the short-
        est path from the destination cell by selecting one of the 4-geometry neighbors
        with the smallest time of arrival value and repeating the selection step by step
        until the source cell is reached.
Step 4: Reversing the path
        Reverse the path derived from step 3 to obtain the desired shortest path.
END {Algorithm of 4-geometry routing}
```

In the 4-geometry router, the wave propagation cannot stop right away when the destination is reached because another path from the same bucket or the next bucket may be a little bit shorter (by less than 1.0). However, if the destination cell is in the current executing bucket, then the algorithm stops since the AT value of the destination cell has been updated to a minimum value that will be proven in theorem 1. Figure 3 explains the detailed computation of the routing algorithm for a single pair. In this illustration, some diagonal paths that cross the tip of the obstacles should be considered as impassable to avoid the ambiguous situation. Figure 3(a) shows the initial grid plane with obstacles that have a source cell S and a destination cell D. After the execution of steps 1 and 2, the AT values of all the cells are obtained as shown in Figure 3(b). Then backtracking and reversing are implemented to obtain the shortest path, as shown in Figure 3(c).

If the 4-geometry routing algorithm is confined to 2-geometry, then it behaves the same as a Lee's algorithm. By comparing the result of Lee's algorithm shown in Figure 4 with that of the 4-geometry router shown in Figure 3(c), the distance from the source to the destination of latter is about 15% shorter.

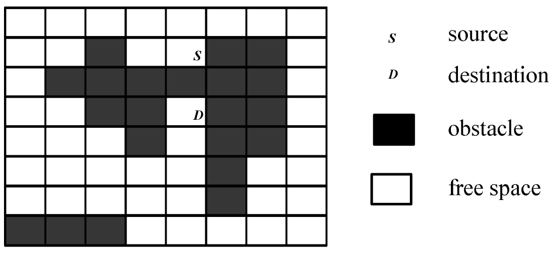
3.3 Performance Analysis of the 4-Geometry Maze Router

The following lemma is applied to prove the correctness of this algorithm, that is, the path obtained from this algorithm is indeed the shortest.

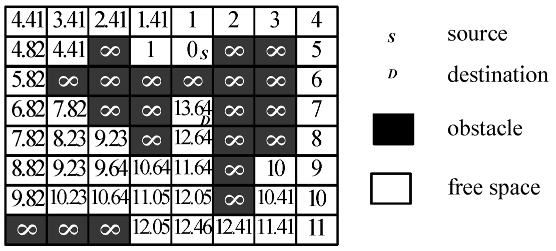
Observation 1. Once all the cells in the LL_{index} have updated the AT values of its neighboring cells and the LL_{index} is empty, all cells in the TL should be moved into either $LL_{(index+1)\bmod 3}$ or $LL_{(index+2)\bmod 3}$ according to their $AT_{i,j}$ values.

According to steps 2.3.2 and 2.3.3, the minimum $AT_{i,j}$ value of cells in the TL would be no less than

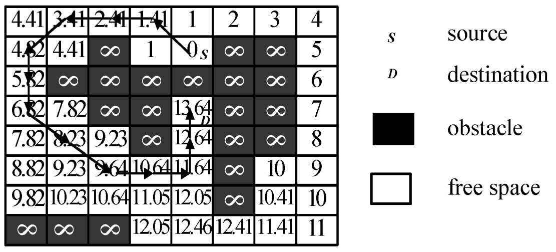
$$(\text{minimum } AT_{i,j} \text{value in the } LL_{index}) + 1$$



(a) The initial grid plane



(b) Execution result of steps 1 and 2



(c) Backtrack and reverse to obtain the shortest path

Fig. 3. Illustration of the 4-geometry routing algorithm on an 8 × 8 grid of cells.

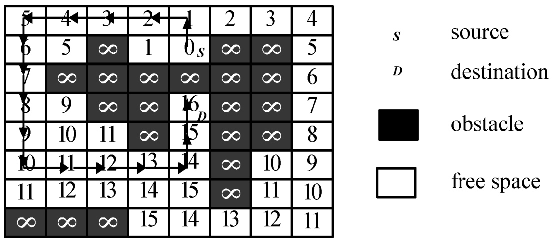


Fig. 4. Illustration of Lee's algorithm on the 8 × 8 grid of cells.

and the maximum $AT_{i,j}$ value of cells in the TL would be no greater than

$$(\text{maximum } AT_{i,j} \text{ value in the } LL_{index}) + \sqrt{2}.$$

The LL_{index} is empty when step 2.3 is completed. After the computation of step 2.4.1, all of the cells in the temporary list TL should be moved into either $LL_{(index+1)\bmod 3}$ or $LL_{(index+2)\bmod 3}$ according to their $AT_{i,j}$ values.

Observation 2. All of the previous buckets should be empty in step 2.3.

LEMMA 1. *The $AT_{i,j}$ values of cells in the LL_{index} are minimal in step* 2.3.

PROOF. This lemma is proven by contradiction. For the $AT_{i,j}$ value of any cell $C_{i,j}$ in the current bucket LL_c, where $0 < c \leq AT_{\max}$, we assume that there exists a smaller $AT'_{i,j}$ value updated by another cell $C_{i',j'}$ that has $AT_{i',j'}$ value. According to step 2.3.2, the minimum $AT_{i,j}$ value of the cell $C_{i,j}$, should be replaced by $AT'_{i,j}$ if the cell $C_{i',j'}$ is in one of the previous buckets LL_l where $0 \leq l \leq c - 1$ and LL_c is the current bucket. If the cell $C_{i',j'}$ is in the current or one of the higher buckets LL_l, where $c \leq l \leq AT_{\max}$, we know that $c \leq AT_{i',j'}$ and $c \leq AT_{i,j} < c + 1$. Also $AT'_{i,j} = AT_{i',j'} + dist$, where the distance between two neighboring cells, *dist*, is either 1 or $\sqrt{2}$, from the definition of the $AT_{i,j}$ value of a cell $C_{i,j}$ in the bucket LL_l. Thus, it can be concluded that $AT_{i,j} < c + 1 \leq AT'_{i,j}$, which is in contradiction to the assumption.

We then conclude that the $AT_{i,j}$ values of cells in the LL_{index} in step 2.3 are minimal. □

THEOREM 1. *The shortest path between any two different cells on a grid plane can be obtained from the algorithm if it exists.*

PROOF. If a source cell is determined, all of the cells surrounding the source cell, except the obstacles, will be moved into, and later removed from, the corresponding bucket exactly once during the computation. According to Lemma 1, all of the $AT_{i,j}$ values of these cells are minimal after the computation. Thus, we can obtain the shortest path from the algorithm. □

LEMMA 2. *Each cell in the grid plane, except the obstacle cell, will not be moved back to the bucket once it is removed.*

PROOF. In this searching algorithm, the indices of cell $C_{i,j}$ are moved into the bucket again if and only if the $AT_{i,j}$ value of the cell is updated to a smaller value. According to Lemma 1, each cell removed from the bucket has the minimum value. Therefore, the $AT_{i,j}$ value of the cell cannot be updated to a smaller value once the indices of cell $C_{i,j}$ are removed from the bucket. They will not be moved back into the bucket again. □

In the following theorem, we prove the time complexity of this algorithm.

THEOREM 2. *The shortest path algorithm has a time complexity of $O(N)$.*

PROOF. Steps 1, 3, and 4 of this searching algorithm have a time complexity of $O(N)$. According to Lemma 2, once the indices of cell $C_{i,j}$ are removed from the bucket, it will not be moved into the bucket again. This means that the number of cells removed from the bucket are no greater than N during the process. Each cell has the time complexity of $O(1)$ to update the AT values of its eight neighboring cells. We therefore know that the time complexity for step 2 is $O(N)$. It is concluded that the shortest path has the time complexity of $O(N)$ from steps 1 to 4. □

The complexity of this algorithm would increase by $O(U/L)$ only if U/L can be normalized, where U and L are the upper and lower bounds for edge weights, since we would need at least $(U/L)N$ number of buckets in that case.

3.4 The Higher-Geometry Maze Router

For the 8-geometry maze router, the 4-geometry router can be modified by simply increasing the number of circular buckets to four and some conditional statements, for the $\sqrt{5}$ step move. Meanwhile, a *Dir* parameter in the cell map is required to keep track of which predecessor causes the minimum *AT* value. The 8-geometry uses at most twice the time as the 4-geometry. Thus, for the 16-geometry, 32-geometry, and other higher-geometry maze routers, it is merely required to increase the number of circular buckets and conditional statements. The running time is $O(\lambda N)$ for the λ-geometry algorithm. The performance analysis of the higher geometry routers is neglected since it is similar to the 4-geometry case.

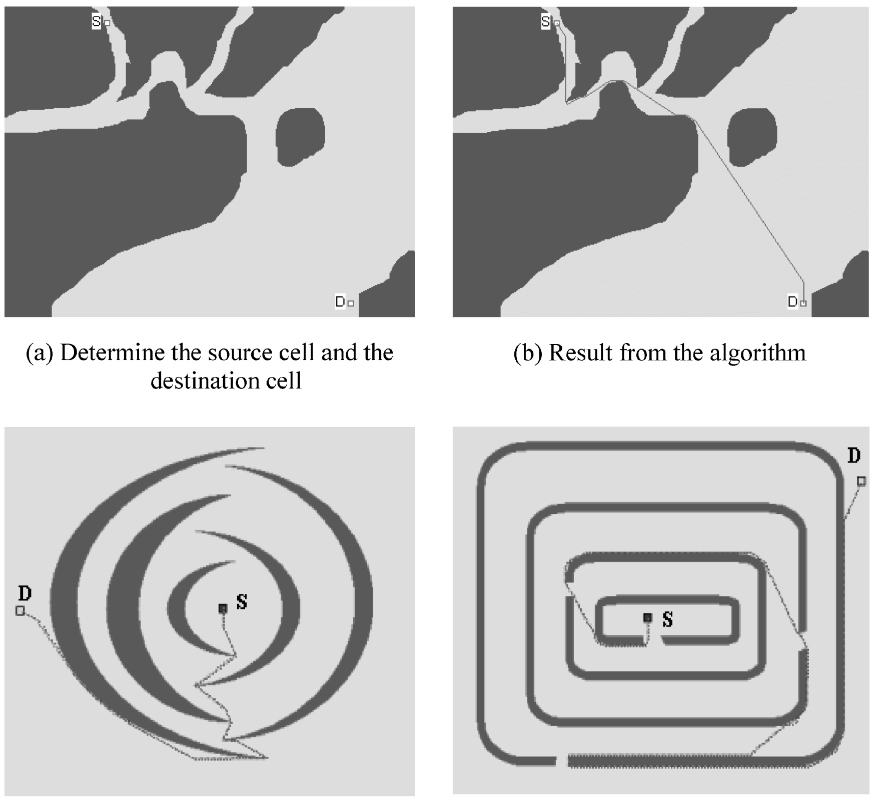
Furthermore, the F/O parameter is introduced for various terrains in which the velocity of the object for those cells in the specific terrains is divided by the value of its F/O parameter. The uniform wave propagation for various terrains can be achieved by increasing a certain number of buckets. As a whole, the algorithm is capable of finding the optimal path in the grid plane among various terrains. If there is only one terrain in the grid plane, the optimal path is also called the *shortest path*.

3.5 Experimental Results

An example of the maze routing on a raster electronic map is illustrated in Figure 5. The source cell S and destination cell D are marked in Figure 5(a). In a 400×300 pixel (considered as cells) raster electronic map, finding the shortest path by the 4-geometry maze router between a source cell and a destination cell takes only a fraction of a second for the Pentium III PC. Figure 5(b) indicates the shortest path presented by a curve. Figures 5(c) and 5(d) are two examples depicted by implementing the 8-geometry router, in which each 45° angle in the 4-geometry has been divided into angles of 26.4° and 18.6°. It is interesting to compare our algorithm with the popular A* algorithm [Hart et al. 1968] used in computer games, but note that the A* algorithm can be trapped in some mazes, whereas our algorithms guarantee finding the shortest path if one exists.

4. ROUTING OF MULTITERMINAL NETS

The problem of finding the minimal-length wire routing for multiterminal nets belongs to the class of Steiner tree problems. The *Steiner tree problem* is a minimum interconnection problem. The problem can also be applied in the geometric realm; the most common variants are the Euclidean Steiner tree and the rectilinear Steiner tree problems. In additional to the Euclidean and rectilinear metrics, octagonal metrics (4-geometry) has been considered. Here, the input is a set of vertices in the plane that are the terminals, and the goal is to obtain a tree of minimum length (in the appropriate metric) that connects all the terminals. Up to now, many exact, heuristic, and genetic algorithms have been proposed



(a) Determine the source cell and the destination cell (b) Result from the algorithm

(c) Crescents pattern (d) Maze pattern

Fig. 5. Illustrations of the maze routing on the raster electronic map.

for constructing these trees [Lee et al. 1976; Hwang 1979; Smith et al. 1981; Dreyer and Overton 1998]. However, only some of these algorithms solve the Steiner tree in a plane with obstacles [Winter, 1993; Ganley and Cohoon, 1994]. The problem of constructing Steiner minimal trees in the Euclidean plane without obstacles is NP-hard. When obstacles are included, the obstacle-avoiding Steiner minimal-trees problem is at least NP-hard as well [Cieslik, 1998]. The main objective of this section is to develop a heuristic algorithm based on the maze routers to construct Steiner trees in 2-geometry and 4-geometry. The algorithm introduces a (global) Torricelli vertex to construct the obstacle-avoiding Steiner trees based on the cost accumulation of the cells. After that, a cost accumulation scheme is also used to improve the Steiner tree locally. The algorithm is presented for the rectilinear case since the exact solution of rectilinear Steiner trees for small instances can be obtained without any difficulty.

4.1 Definition of the Torricelli Vertex

For multiterminal nets problems, the maze routing approach is one of the basic algorithms. The traditional maze router connects the first two terminals, and then the entire connected path is the target for the wave propagation from the third terminal, etc. Thus, all the r terminals of $Z = (Z_1, \ldots, Z_r)$ are considered as the sources for the wave propagation to the other terminals at least once. The quality of its result strongly depends on the ordering of the terminals since it is difficult to determine the optimal terminal ordering. In order to improve the routing quality of multiterminal nets, a Torricelli vertex (point), a vertex that has the minimum total cost to all terminals, is introduced for further routing. The definition of this vertex can be considered as the following geometric problem:

Given Z, where terminal i is at location (x_i, y_i), find a vertex (x_c, y_c) such that $D_c = \sum_{1 \le i \le r} d_i$ is minimum, where, d_i is the distance from the ith vertex (x_i, y_i) to (x_c, y_c), and r is the total number of vertices. The vertex (x_c, y_c) has the minimum total cost connecting to those r terminals including superposition connection. The cost (length) of the near optimal Steiner tree is no more than the value of the Torricelli vertex since the superposition (overlap) path will be removed after the routing process. If the maze router uniformly propagates from each source vertex to the remaining vertices, then the cost (time of arrival) of all vertices in the cell map is specified. The cells with equal cost in the grid plane can be considered as a level curve, which is similar to the sea-level curve in a geographical map. A solution of this problem is to obtain a vertex that minimizes the sum of the cost from all of the r terminals. Thus, the minimization of the above problem can be formulated as the following algebraic program:

Minimize $\sum_{1 \le i \le r} d_i$
subject to $(x_c - x_i)^2 + (y_c - y_i)^2 \le d_i^2, i = 1, \ldots, r,$

where $d_i \ge 0$ and r is the total number of terminals.

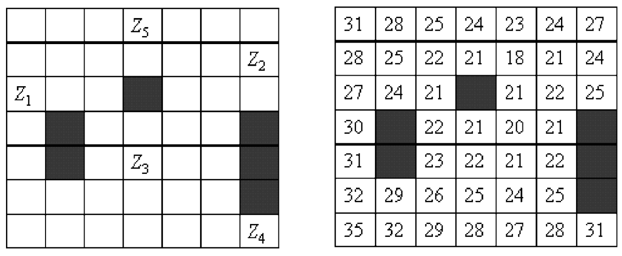
This is Fermat's problem in a Euclidean plane [Brimberg 1995], which is the local version of Steiner's problem. If the r terminals are not collinear in a Banach-Minkowski space $M_d(B)$ and if B is a strictly convex unit ball, then the Torricelli vertex (Steiner vertex) is a single vertex. A Banach-Minkowski space is uniquely defined by a d-dimensional affined space A_d and convex and compact body B. Usually, a (finitely or infinitely dimensional) linear space, which is complete with regard to its given norm, is called *Banach space*. This is a natural idea to construct a 1-Steiner (Torricelli) as an approximation of an SMT (Steiner minimal tree). Instead of using the iterative Weiszfeld's algorithm [Kuhn 1973] in the above algebraic program, our algorithm introduces a cost accumulation scheme to obtain the Torricelli vertex.

4.2 An Algorithm for Near Optimal Rectilinear Steiner Trees

For rectilinear Steiner trees (RSTs), Fermat's problem can be described as follows:

Minimize $\sum_{1 \le i \le r} |x_c - x_i| + |y_c - y_i|$, where (x_c, y_c) has the minimum total rectilinear cost to all terminals.

The goal of our algorithm is to find a near optimal Steiner tree starting from (x_c, y_c) to connect all the terminals. The minimum rectilinear Steiner tree



31	28	25	24	23	24	27
28	25	22	21	18	21	24
27	24	21	▨	21	22	25
30	▨	22	21	20	21	▨
31	▨	23	22	21	22	▨
32	29	26	25	24	25	▨
35	32	29	28	27	28	31

(a) Initial state (b) Cost accumulation table from Z_1 to Z_5

Fig. 6. Example of the cost accumulation by using the maze routing algorithm.

problem in the plane is defined as follows: given a set of r terminals in an $m \times n$ grid plane of N vertices, a set S_t of Steiner vertices such that the spanning tree over $Z \cup S_t$ has minimum cost is found.

The cost accumulation scheme presented in this article is to obtain the Torricelli vertex for further routing and local refinement of multiterminal nets. The way to find the set of Torricelli vertices is still unknown, but a single global Torricelli vertex can be obtained by the cost accumulation of the r terminals in this algorithm. For instance, there are five terminals that are designed to connect to an RST, as shown in Figure 6(a). Thus, the five cost tables from Z_1 to Z_5 are generated individually. The cost accumulation table is the sum of costs in the five tables, as shown in Figure 6(b), where the Torricelli vertex is the vertex with the minimum value in the cost accumulation table. Note that the cost accumulation scheme uses extra memory space to reduce the time complexity.

Once the cost accumulation scheme is applied in the grid plane, the algorithm starts from the Torricelli vertex to approximate a near optimal Steiner tree. First of all, we can obtain the nearest terminal Z_i from the Torricelli vertex, and then the first critical path is connected from Z_i to its nearest terminal Z_j, where Z_i and Z_j belong to Z. After that, the entire connected paths are considered as the targets to the remaining terminals. Based on the sum of costs in the remaining terminal tables, the next critical path is a shortest path between a nearest terminal and the existing (entire) connected paths, and the new entire connected paths are then the combination of this new critical path and the previous entire connected paths. Continue these steps until all terminals are connected. Also, a critical vertex is defined as the connection vertex between the shortest path and the entire connected paths.

For the local refinement, the same cost accumulation scheme is applied locally if the localized subset is determined. The local refinement selects subsets of three or four connected vertices by using the method of the Steiner visibility region [Winter 1993]. In order to search and locate which vertices are most likely to appear near each other in the tree, the beam of light originating from the global Torricelli's vertex is rotated all the way around in the grid plane. The local Torricelli (Steiner) vertex connects these three or four vertices and the Steiner tree replaces the original path if the rerouted Steiner tree has a smaller length (cost) than the original one. Also, it should be noted that the cost accumulation scheme starts from a Torricelli vertex and may induce several Steiner vertices by using the maze router to connect the multiterminal nets. The algorithm can be summarized in the following steps:

Heuristic algorithm:

Step 1: Individual costs
For each terminal in Z, a wave propagates to the remaining vertices in the cell map and stores its costs in the corresponding cost table. (These r cost tables generated by the r terminals are stored and used for the rest steps of our algorithm.)

Step 2: Cost accumulation
A cost accumulation table is generated based on the sum of the r cost tables. The vertex with minimum total cost in the cost accumulation table is considered as a Torricelli vertex for further constructing the SMT.

Step 3: Initial critical path
Obtain the nearest terminal Z_i from the Torricelli vertex by an index search in the r tables to find the minimum cost between all the r terminals and the Torricelli vertex. The first critical path of multiterminal nets is obtained from the Z_i that backtracks to its nearest terminal Z_j, where Z_i and Z_j belong to Z.

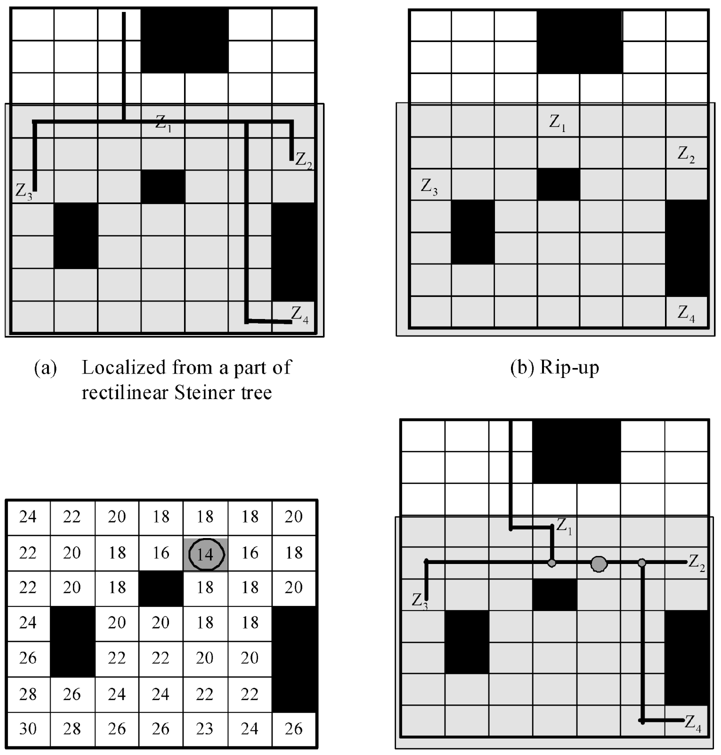
Step 4: Iterations
The entire connected paths are considered as the target vertices to the remaining terminals. Once the critical vertex is determined, backtrack to the nearest terminal to obtain the shortest path and then insert it into the entire connected paths. For the purpose of shortening the sum of the lengths in the remaining critical paths, choose the optimal one according to the minimum sum of costs in the remaining terminal tables whenever two or more shortest paths exist. Both the critical vertex and its nearest terminal can be found by an index search in the $r - 1$ tables with sentinel data structure. Repeat this step until all of the terminals are connected. The final obtained connected paths are the near optimal Steiner tree for the given r terminals.

Step 5: Local refinement
Identify subsets of three or four terminals of Z, which are most likely to appear near each other in the tree by the method of the Steiner visibility region. The local Torricelli vertices are obtained by the cost accumulation scheme of these subsets. From these vertices, the cost accumulation scheme, as in steps 3 and 4, is applied to obtain the local minimal Steiner tree as shown in Figure 7, in which the particular local Steiner tree has four terminals. Compare the rerouted path with the local tree (connected paths) in this subset and replace the local tree if the new tree has a smaller length. Repeat this process for the whole Steiner tree.

The space and time complexities of the proposed heuristic algorithm are easily analyzed and can be obtained as $O(rN)$ and $O(r^2N)$, respectively.

4.3 Exact Algorithm of Minimum Rectilinear Steiner Trees

Corresponding to multiterminal routing, the rectilinear Steiner minimum tree problem cannot be solved exactly by any polynomial time algorithms. Nonetheless, exponential time algorithms have been devised to solve the SMT problem exactly for small instances in a reasonable time. Here, an exact algorithm is computed to compare the heuristic result. The exact algorithm for the SMT problem is an exhausted searching method. Once the numbers and locations of the terminals and obstacles are given, the algorithm runs all the possible combination for free vertices if all of the terminals can be connected. To reduce



24	22	20	18	18	18	20
22	20	18	16	14	16	18
22	20	18		18	18	20
24		20	20	18	18	
26		22	22	20	20	
28	26	24	24	22	22	
30	28	26	26	23	24	26

(a) Localized from a part of rectilinear Steiner tree

(b) Rip-up

(c) Obtain the Torricelli point (Steiner point) from the cost accumulation table

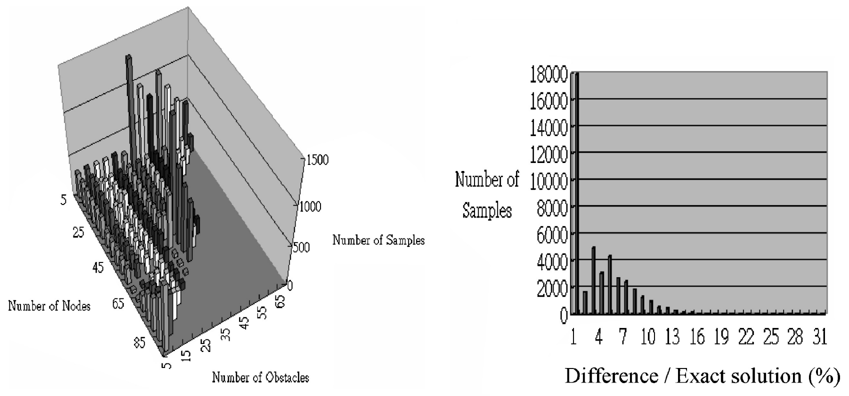
(d) Result of the cost accumulation scheme

Fig. 7. An example of the cost accumulation scheme in step 5.

the overall computation, an upper bound is defined as the length obtained from the heuristic algorithm and the number of terminals minus 1 is defined as a lower bound.

4.4 Experimental Results

We have analyzed the performance of this algorithm for the examples of multi-terminal nets. The given parameters within a square region (e.g., 10×10, and some of 20×20) are the samples that include obstacles and specified r terminals. The locations of terminals and obstacles are randomly generated. For the given numbers of terminals and obstacles (also called a *set of samples*), if the set of samples runs all of its combination in the grid plane for more than 5 h then it is discarded. According to the density of the terminals (nodes) and obstacles, we have 42,890 effective samples, as shown in Figure 8(a), that run both heuristic and exact algorithms. The distribution of length differences between the heuristic and exact solutions is shown in Figure 8(b). From the statistical criteria, we first calculate the difference of each sample by$X_i = length_{heuristic} - length_{exact}$, where $0 \leq i < n$ and n is the number of the samples. After that, we obtain the mean, $\overline{X} = \sum_{i=0}^{n-1} X_i/n$, and calculate the confidence interval *CI* by the formulation of the statistical criteria $CI = \pm 1.96(\overline{X}/\sqrt{n})$. Next, we calculate 95% of



(a) Distribution of the samples (b) Distribution of the difference (%)

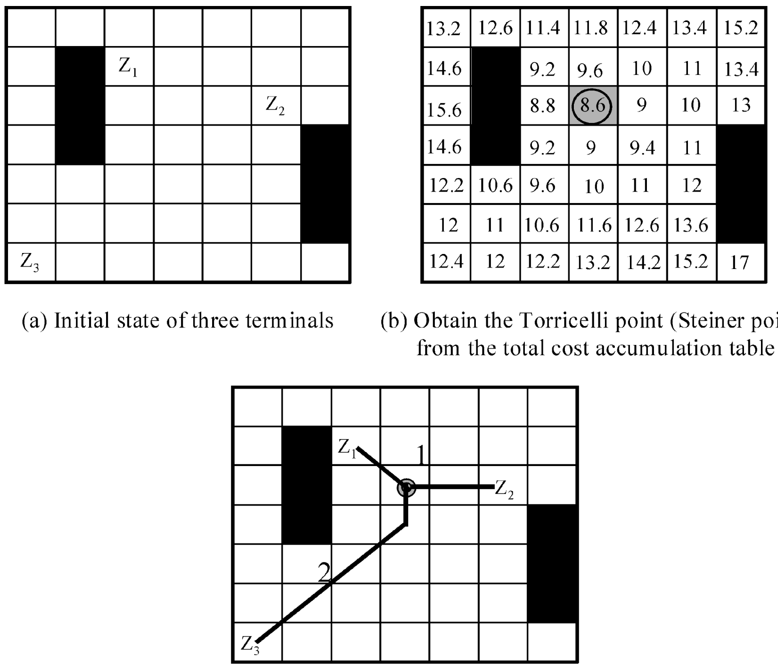
Fig. 8. Distribution of the samples and their differences with exact solutions.

$CI/\overline{X} = 0.0195 \leq 0.025$ to determine whether the sample is reliable. The standard deviation is 1.34, and that means the standard deviation value between the heuristic and exact algorithms is between 0–1.34 units of length. Note that the heuristic solutions have a larger difference if the number of the terminals is large, since it is more like a k-Steiner tree problem. We present the empirical evidence that this algorithm implies a good solution for multiterminal nets in the obstacle-avoiding Steiner tree problem. For example, the worst case is bounded by 2.52 units of length for a 10×10 grid plane.

4.5 4-Geometry Steiner Trees

Recently, nonrectilinear geometry has gained ground because of enhanced computing capabilities and the need for the design of high-performance circuits. Our heuristic algorithm can be extended to 4-geometry since it was designed to find near optimal Steiner trees in the grid plane. The procedures of constructing the 4-geometry Steiner tree are similar to those for the rectilinear Steiner tree except that the diagonal direction is included in the backtracking step.

After the Torricelli point is determined from the cost accumulation table, as shown in Figure 9, we can obtain its nearest terminal, Z_1. The constructing of critical path 1 backtracks from Z_1 to its nearest terminal, Z_2. The constructing of critical path 2 backtracks from the vertices in path 1 to Z_3, etc. An example in a 20×20 grid plane that has 12 connecting terminals with components (obstacles) is presented in Figure 10 to demonstrate the difference between the rectilinear and 4-geometry Steiner trees. Figure 10(a) is the experimental result, showing the length of the rectilinear tree to be 66 units, where the squares filled with black represent the circuit components considered as the obstacles and the circles represent the connecting terminals. Also, this experimental result has the same length as the exact solution, taking 0.12 s on a Pentium III PC with 256 MB of memory. If the 4-geometry maze router is applied to this sample, the length is 58.87 units, as shown in Figure 10(b), and is nearly reduced by 11% compared to the rectilinear case. It is time-consuming for the 4-geometry to get the exact solution; only some small nets (10×10) were tested, and the



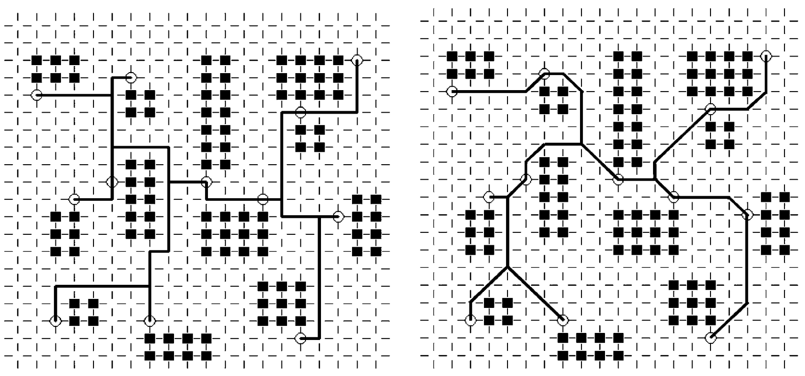
13.2	12.6	11.4	11.8	12.4	13.4	15.2
14.6	■	9.2	9.6	10	11	13.4
15.6	■	8.8	8.6	9	10	13
14.6	■	9.2	9	9.4	11	■
12.2	10.6	9.6	10	11	12	■
12	11	10.6	11.6	12.6	13.6	■
12.4	12	12.2	13.2	14.2	15.2	17

(a) Initial state of three terminals

(b) Obtain the Torricelli point (Steiner point) from the total cost accumulation table

(c) Order of path construction and result of a 4-geometry Steiner tree

Fig. 9. Example of constructing a 4-geometry Steiner tree.



(a) A rectilinear Steiner tree (b) A 4-geometry Steiner tree

Fig. 10. Illustration of rectilinear and 4-geometry Steiner trees.

results showed the length difference between the heuristic and exact solutions to be around 0.67% in the 1000 test samples. The worst case had 2.8 units of length difference in one sample.

However, the difference in length reduction was quite marginal for the 4-geometry and the 8-geometry routings. Also, the terminal ordering problem may increase the length slightly in the higher-geometry routings compared to the 4-geometry routing. Thus, it is effective enough to run the 4-geometry routing for most of the SMT problems with obstacles. The main difference between our router and other 4-geometry routers is that our algorithm can work on various terrains and also work on a geometry value higher than 4 (i.e., 8-geometry, 16-geometry, etc.), which is useful for other research areas.

5. CONCLUSIONS

This article presents a 4-geometry maze routing algorithm and its application to Steiner tree problems. In this study, we improved the traditional maze router by properly using the data structure of buckets, a simple temporary liked-list, TL, cell matrices and indices (i, j) (sequentially allocated the memory in constant time) with linear space to avoid any sorting or presorting. In regard to 4-geometry, we were able to significantly shorten the optimal path between a source cell and a destination cell. Our new algorithm has both $O(N)$ time and space complexities since we did not sort or presort the edge weights at all. Without any difficulty, the algorithm is capable of being extended to 8-geometry, 16-geometry, etc., for use in other research areas such as the path planning of robot motion, raster charts, or nondistorted digital maps, etc. For the various terrains problem, the algorithm is able to find an optimal path between a given pair of cells by modifying the number of required buckets.

For the multiterminals net, the cost accumulation scheme was applied to the grid-based Steiner tree problems, in which the Torricelli vertices are located, to improve the routing. The experimental results showed that the algorithm works well in the RST problems. In addition, using the 4-geometry routing, path lengths can be reduced around 10–12% compared to those in the 2-geometry routing. But if the geometry value is higher than 4, the gain for the obstacle-avoiding Steiner tree is quite marginal.

The proposed method can be extended from two-dimensional grid planes to three-dimensional volumes and from static systems to dynamic (time-varying) systems by adding spatial and time parameters, respectively. It also can be applied to multilayer global routing problems.

REFERENCES

Ahuja, R. K., Mehlhorn, K., Orlin, J. B., and Tarjan, R. E. 1990. Faster algorithms for the shortest path problem. *J. Assoc. Comput. Mach. 37*, 2 (April), 213–223.

Barraquand, J. and Latombe, J. C. 1991. Robot motion planning: A distributed representation approach. *Int. J. Robotics Res. 10*, 6 (Dec.), 628–649.

Brimberg, J. 1995. The Fermat-Weber location problem revisited.*Math. Programm. B 71*, 1, 71–76.

Cieslik, D. 1991. The 1-Steiner-minimal-tree problem in Minkowski-spaces, *Optimization 22*, 2, 291–296.

Cieslik, D. 1998. *Steiner Minimal Trees*, Kluwer Academic Publishers, Dordrecht, The Netherlands.

Dí-az de León S., J. L., and Sossa A., J. H. 1998. Automatic path planning for a mobile robot among obstacles of arbitrary shape. *IEEE Trans. Syst. Man. Cybernet. B. 28*, 3 (June), 467–472.

Dijkstra, E. W. 1959. A note on two problems in connexion with graphs. *Numer. Math. 1*, 269–271.

Dinic, E. A. 1978. Economical algorithms for finding shortest paths in a network, In *Transportation Modeling Systems*, Y. S. Popkov and B. L. Shmulyian, Eds. Institute for System Studies, Moscow, Russia, 36–44.

DREYER, D. R. AND OVERTON, M. L. 1998. Two heuristics for the Euclidean Steiner tree problem. *J. Glob. Opt. 13*, 1, 95–106.

FAWCETT, J. AND ROBINSON, P. 2000. Adaptive routing for road traffic. *IEEE Comput. Graph. Appl. 20*, 3 (May/June), 46–53.

GANLEY, J. L. AND COHOON, J. P. 1994. Routing a multi-terminal critical net: Steiner tree construction in the presence of obstacles. *Proceedings of the IEEE International Symposium on Circuits and Systems*, vol. 1. 113–116.

GEORGAKOPOLULOS, G. AND PAPADIMITRIOUS, C. H. 1987. The 1-Steiner tree problem. *J. Algorith. 8*, 1, 122–130.

GOLDBERG, A. V. 2001. Simple shortest path algorithm with linear average time. *Proceedings of the European Symposium on Algorithms (ESA)*. Lecture Notes in Computer Science, Vol. 2161. Springer-Verlag, Berlin Germeny, 230–241.

HART, P. E., NILSSON, N. J., and RAPHAEL, B. 1968. A formal basis for the heuristic determination of minimum cost paths. *IEEE Trans. Syst. Sci. Cybernet. 4*, 2, 100–107.

HENZINGER, M. R., KLEIN, P., RAO, S., AND SUBRAMANIAN, S. 1997. Faster shortest-path algorithms for planar graphs. *J. Comput. Syst. Sci. 55*, 1 (Aug.), 3–23.

HOEL, J. H. 1976. Some variations of Lee's algorithm. *IEEE Trans. Comput. C-25*, 1 (Jan.), 19–24.

HWANG, F. K. 1979. An $O(n \log n)$ algorithm for suboptimal rectilinear Steiner trees. *IEEE Trans. Circ. Syst. CAS-26*, 75–77.

JAN, G. E., LIN, M. B., AND CHEN, Y. Y. 1997. Computerized shortest path searching for vessels. *J. Marine Sci. Tech. 5*, 1, 95–99.

KAHNG, A. AND ROBINS, G. 1992. A new class of iterative Steiner tree heuristics with good performance.*IEEE Trans. Comput.-Aided Des. Integrat. Circ. Syst. 11*, 7, 893–902.

KIMMEL, R., AMIR, A., AND BRUCKSTEIN, A. M. 1995. Finding shortest paths on surfaces using level sets propagation. *IEEE Trans. Patt. Anal. Mach. Intell. 17*, 6 (June), 635–640.

Kuhn, H. W. 1973. A note on Fermt's problem. *Math. Programm. 4*, 98–107.

LEE, C. Y. 1961. An algorithm for path connections and its applications. *IRE Trans. Electron. Comput. EC-10*, Sept., 346–365.

LEE, J. H., BOSE, N. K., AND HWANG, F. K. 1976. Use of Steiner's problem in suboptimal routing in rectilinear metric. *IEEE Trans. Circ. Syst. CAS-23*, 470–476.

LIN, Y. L., HSU, Y. C., AND TSAI, F. S. 1990. Hybrid routing. *IEEE Trans. Comput.-Aided Des. Integr. Circ. Syst. 9*, 2 (Feb.), 151–157.

OGNIEWICZ, R. L. AND KUBLER, O. 1995. Voronoi tessellation of points with integer coordinates: Time-efficient implementation and online edge-list generation. *Patt. Recogn. 28*, 12 (Dec.), 1839–1844.

RUBIN, F. 1974. The Lee path connection algorithm. *IEEE Trans. Comput. C-23*, 9 (Sept.), 907–914.

SHERWANI, N. A. 1999. *Algorithms for VLSI Physical Design Automation*, 3rd. ed. Kulwer Academic Publishers, Boston, MA, 260–279.

SMITH, J. M.-G., LEE, D. T., AND LIEBMAN, J. S. 1981. An $O(n \log n)$ heuristic for Steiner minimal tree problems on the Euclidean metric. *Networks 11*, 1, 23–39.

TEIG, S. L. 2002. The X architecture: Not your father's diagonal wiring. In *Proceedings of the 2002 International Workshop on System-Level Interconnect Prediction* (San Diego, CA, Apr. 6–7), ACM Press, New York, NY, 33–37.

THORUP, M. 1999. Undirected single-source shortest paths with positive integer weights in linear time. *J. Assoc. Comput. Mach. 46*, 3, 362 – 394.

WINTER, P. 1993. Euclidean Steiner minimal trees with obstacles and Steiner visibility graphs. *Discrete App. Math. 47*, 2, 187–206.

XING, Z. AND KAO, R. 2002. Shortest path search using tiles and piecewise linear cost propagation. *IEEE Trans. Comput.-Aided Des. Integr. Circ. Syst. 21*, 2 (Feb.), 145–158.

Received June 2003; revised October 2003, December 2003; accepted December 2003

Algorithmic Aspects of Hardware/Software Partitioning

PÉTER ARATÓ, ZOLTÁN ÁDÁM MANN, and ANDRÁS ORBÁN
Budapest University of Technology and Economics

One of the most crucial steps in the design of embedded systems is hardware/software partitioning, that is, deciding which components of the system should be implemented in hardware and which ones in software. Most formulations of the hardware/software partitioning problem are $\mathcal{NP}$-hard, so the majority of research efforts on hardware/software partitioning has focused on developing efficient heuristics.

This article considers the combinatorial structure behind hardware/software partitioning. Two similar versions of the partitioning problem are defined, one of which turns out to be $\mathcal{NP}$-hard, whereas the other one can be solved in polynomial time. This helps in understanding the real cause of complexity in hardware/software partitioning. Moreover, the polynomial-time algorithm serves as the basis for a highly efficient novel heuristic for the $\mathcal{NP}$-hard version of the problem. Unlike general-purpose heuristics such as genetic algorithms or simulated annealing, this heuristic makes use of problem-specific knowledge, and can thus find high-quality solutions rapidly. Moreover, it has the unique characteristic that it also calculates *lower bounds on the optimum solution*. It is demonstrated on several benchmarks and also large random examples that the new algorithm clearly outperforms other heuristics that are generally applied to hardware/software partitioning.

Categories and Subject Descriptors: J.6 [**Computer-Aided Engineering**]: *Computer-aided design (CAD)*; F.2.2 [**Analysis of Algorithms and Problem Complexity**]: Nonnumerical Algorithms and Problems; G.2.1 [**Discrete Mathematics**]: Combinatorics—*Combinatorial algorithms*

General Terms: Algorithms, Design

Additional Key Words and Phrases: Hardware/software partitioning, hardware/software codesign, graph bipartitioning, graph algorithms, optimization

1. INTRODUCTION

Today's computer systems typically consist of both hardware and software components. For instance, in an embedded signal processing application it is common to use both application-specific hardware accelerator circuits and

Partial support of the Hungarian National Science Fund (Grant Nos. OTKA T043329 and T042559) is gratefully acknowledged.
Authors' addresses: Department of Control Engineering and Information Technology, Budapest University of Technology and Economics, H-1117 Budapest, Magyar tudósok körútja 2, Hungary; email: arato@iit.bme.hu, {zoltan.mann,andras.orban}@cs.bme.hu.

general-purpose, programmable units with the appropriate software [Arató et al. 2003b].

This is beneficial since application-specific hardware is usually much faster than software, and also more power-efficient, but it is also significantly more expensive. Software, on the other hand, is cheaper to create and to maintain, but slow, and general-purpose processors consume much power. Hence, performance- or power-critical components of the system should be realized in hardware, and noncritical components in software. This way, an optimal tradeoff between cost, power, and performance can be achieved.

One of the most crucial steps in the design of such systems is partitioning, that is, deciding which components of the system should be realized in hardware and which ones in software. Clearly, this is the step in which the above-mentioned optimal tradeoff should be found. Therefore, partitioning has dramatic impact on the cost and performance of the whole system [Mann and Orbán 2003]. The complexity of partitioning arises because conflicting requirements on performance, power, cost, chip size, etc., have to be taken into account.

Traditionally, partitioning was carried out manually. However, as the systems to design have become more and more complex, this method has become infeasible, and many research efforts have been undertaken to automate partitioning as much as possible.

1.1 Previous Work

Based on the partitioning algorithm, exact and heuristic solutions can be differentiated. The proposed exact algorithms include branch-and-bound [Binh et al. 1996], dynamic programming [Madsen et al. 1997; O'Nils et al. 1995], and integer linear programming [Mann and Orbán 2003; Niemann 1998; Niemann and Marwedel 1997].

The majority of the proposed partitioning algorithms are heuristic. This is due to the fact that partitioning is a hard problem, and, therefore, exact algorithms tend to be quite slow for bigger inputs. More specifically, most formulations of the partitioning problem are $\mathcal{NP}$-hard, and the exact algorithms for them have exponential runtimes. The $\mathcal{NP}$-hardness of hardware/software partitioning was claimed by several researchers [Binh et al. 1996; Eles et al. 1996; Kalavade 1995; Vahid and Gajski 1995], but we know only about one proof [Kalavade 1995] for a particular formulation of the hardware/software partitioning problem.

Many researchers have applied general-purpose heuristics to hardware/software partitioning. In particular, genetic algorithms have been extensively used [Arató et al. 2003a; Dick and Jha 1998; Mei et al. 2000; Quan et al. 1999; Srinivasan et al. 1998], as well as simulated annealing [Eles et al. 1997; Ernst et al. 1993; Henkel and Ernst 2001; Lopez-Vallejo et al. 2000]. Other, less popular heuristics in this group are tabu search [Eles et al. 1997] and greedy algorithms [Chatha and Vemuri 2001; Grode et al. 1998].

Some researchers used custom heuristics to solve hardware/software partitioning. This includes the GCLP algorithm [Kalavade and Lee 1997; Kalavade and Subrahmanyam 1998] and the expert system of Lopez-Vallejo and Lopez [1998, 2003], as well as the heuristics by Gupta and de Micheli [1993] and Wolf [1997].

There are also some families of well-known heuristics that are usually applied to partitioning problems. The first such family of heuristics is hierarchical clustering [Abdelzaher and Shin 2000; Barros et al. 1993; Vahid 2002; Vahid and Gajski 1995]. The other group of partitioning-related heuristics is the Kernighan-Lin heuristic [Kernighan and Lin 1970], which was substantially improved by Fiduccia and Mattheyses [1982], and later by many others [Dasdan and Aykanat 1997; Saab 1995]. These heuristics have been found to be appropriate for hardware/software partitioning as well [Lopez-Vallejo and Lopez 2003; Vahid 1997; Vahid and Le 1997].

Concerning the system model, further distinctions can be made. In particular, many researchers consider scheduling as part of partitioning [Chatha and Vemuri 2001; Dick and Jha 1998; Kalavade and Lee 1997; Lopez-Vallejo and Lopez 2003; Mei et al. 2000; Niemann and Marwedel 1997], whereas others do not [Eles et al. 1996; Grode et al. 1998; Madsen et al. 1997; O'Nils et al. 1995; Vahid and Le 1997; Vahid 2002]. Some even include the problem of assigning communication events to links between hardware and/or software units [Dick and Jha 1998; Mei et al. 2000].

In a number of related articles, the target architecture was supposed to consist of a single software and a single hardware unit [Eles et al. 1996; Grode et al. 1998; Gupta and de Micheli 1993; Henkel and Ernst 2001; Lopez-Vallejo and Lopez 2003; Madsen et al. 1997; Mei et al. 2000; O'Nils et al. 1995; Qin and He 2000; Srinivasan et al. 1998; Stitt et al. 2003; Vahid and Le 1997], whereas others do not impose this limitation. Some limit parallelism inside hardware or software [Srinivasan et al. 1998; Vahid and Le 1997] or between hardware and software [Henkel and Ernst 2001; Madsen et al. 1997]. The system to be partitioned is generally given in the form of a task graph, or a set of task graphs, which are usually assumed to be directed acyclic graphs describing the dependencies between the components of the system.

1.2 Our Approach

In this article, we take a more theoretical approach than most previous works by focusing only on the algorithmic properties of hardware/software partitioning. In particular, we do not aim at partitioning for a given architecture, nor do we present a complete codesign environment. Rather, we restrict ourselves to the problem of deciding—based on given cost values—which components of the system to implement in hardware and which ones in software. This problem will be formalized as a graph bipartitioning problem. Using the graph-theoretic properties of the problem, we can develop more powerful algorithms—as will be shown later. Furthermore, the underlying problem definition is general enough so that the algorithms we propose can be used in many practical cases.

Our aims are the following:

—Clarifying complexity issues, such as: Is partitioning really $\mathcal{NP}$-hard? When is it $\mathcal{NP}$-hard? Why is it $\mathcal{NP}$-hard?

—Developing more powerful partitioning algorithms by capturing the *combinatorial structure* behind the partitioning problem. That is, instead of applying general-purpose heuristics to hardware/software partitioning, we develop algorithms based on the graph-theoretic properties of partitioning. This way, we hope to obtain more scalable algorithms.

Scalability is a major concern when applying general-purpose heuristics. Namely, in order to be fast, such heuristics evaluate only a small fraction of the search space. As the size of the problem increases, the search space grows exponentially (there are 2^n different ways to partition n components), which means that the ratio of evaluated points of the search space must decrease rapidly, leading to worse results. This effect can be overcome only if the small evaluated region contains high-quality solutions. This is exactly what we intend to achieve by making use of the combinatorial properties of the problem.

Specifically, we define two slightly different versions of the hardware/software partitioning problem. One of them is proven to be $\mathcal{NP}$-hard, whereas a polynomial-time exact algorithm is provided for the other one. We believe that this difference sheds some light on the origins of complexity of hardware/software partitioning.

Our main contribution is a novel heuristic algorithm for the $\mathcal{NP}$-hard version of the partitioning problem which is based on the polynomial-time algorithm for the other version of the problem. This heuristic has the property mentioned above that it only evaluates points of the search space that have a high quality in some sense. Consequently, this heuristic outperforms conventional heuristics, which is demonstrated with empirical tests on several benchmarks. Moreover, the new heuristic has the unique property that it can determine a *lower bound on the cost of the optimum solution*, and therefore it can estimate how far the result it found so far lies from the optimum. This is a feature that no previous partitioning algorithm possessed.

The rest of the article is organized as follows. Section 2 provides formal definitions for the hardware/software partitioning problem. This is followed by the analysis of the defined problems in Section 3 and the description of our algorithms in Section 4. Empirical results are given in Section 5, and Section 6 concludes the article. Finally, the proof of our theorems are presented in the Appendix.

2. PROBLEM DEFINITION

2.1 Basic Model

In the basic model the system to be partitioned is described by a *communication graph*, the nodes of which are the components of the system that have to be mapped to either hardware or software, and the edges of which represent communication between the components. Unlike in most previous works, it is not assumed that this graph is acyclic in the directed sense. The edges are not even directed, because they do not represent data flow or dependency. Rather, their role is the following: if two communicating components are mapped to different contexts (i.e., one to hardware and the other to software, or vice versa), then their communication incurs a communication penalty, the value of which is given for each edge as an edge cost. This is assumed to be independent of the direction of the communication (whether from hardware to software or vice versa). If the communication does not cross the hardware/software boundary, it is neglected.

Similarly to the edge costs mentioned above, each vertex is assigned two cost values called *hardware cost* and *software cost*. If a given vertex is decided to be in hardware, then its hardware cost is considered; otherwise its software cost is considered. We do not impose any explicit restrictions on the semantics of hardware costs and software costs; they can represent any cost metrics, like execution time, size, or power consumption. Likewise, no explicit restriction is imposed on the semantics of communication costs. Nor do we impose explicit restrictions on the granularity of partitioning (i.e., whether nodes represent instructions, basic blocks, procedures, or memory blocks). However, we assume that the total hardware cost with respect to a partition can be calculated as the sum of the hardware costs of the nodes that are in hardware, and, similarly, the software cost with respect to a partition can be calculated as the sum of the software costs of the nodes that are in software, just as the communication cost with respect to a partition can be calculated as the sum of the edge costs of those edges that cross the boundary between hardware and software.

While this assumption of additivity of costs is not always appropriate, many important cost factors do satisfy it. For example, power consumption is usually assumed to be additive, implementation effort is additive, execution time is additive for a single processor (and a multiprocessor system can also be approximated by an appropriately faster single-processor system), and even hardware size is additive under suitable conditions [Madsen et al. 1997]. Furthermore, although it is a challenging problem how the cost values can be obtained, it is beyond the scope of this article. Rather, we focus only on algorithmic issues in partitioning.

We now formalize the problem as follows. An undirected graph $G = (V, E)$, $V = \{v_1, \dots, v_n\}$, $s, h : V \to \mathbb{R}^+$, and $c : E \to \mathbb{R}^+$ are given. $s(v_i)$ (or simply s_i) and $h(v_i)$ (or h_i) denote the software and hardware cost of node v_i, respectively, while $c(v_i, v_j)$ (or c_{ij}) denotes the communication cost between v_i and v_j if they are in different contexts. P is called a *hardware-software partition* if it is a bipartition of V: $P = (V_H, V_S)$, where $V_H \cup V_S = V$ and $V_H \cap V_S = \emptyset$. ($V_H = \emptyset$ or $V_S = \emptyset$ is also possible.) The set of crossing edges of partition P is defined as $E_P = \{(v_i, v_j) : v_i \in V_S, v_j \in V_H \text{ or } v_i \in V_H, v_j \in V_S\}$. The hardware cost of P is $H_P = \sum_{v_i \in V_H} h_i$; the software cost of P is $S_P = \sum_{v_i \in V_S} s_i$; the communication cost of P is $C_P = \sum_{(v_i,v_j) \in E_P} c(v_i, v_j)$.

Thus, a partition is characterized by three metrics: its hardware cost, its software cost, and its communication cost. These are rather abstract and possibly conflicting cost metrics that should be optimized together. The most common approach is to assemble a single objective function $f(cost_1, \dots, cost_l)$ containing all the metrics. We consider two versions of f that can be regarded as the two extremes—yielding two rather different versions of the partitioning problem.

In the first version, the aim of partitioning is to minimize the weighted sum of these three metrics, that is, f is linear in all of its arguments. The weights are specified by the designer, and define the relative importance of the three metrics. More formally, we define the total cost of P as $T_P = \alpha H_P + \beta S_P + \gamma C_P$, where α, β, and γ are given nonnegative constants, and the aim is to minimize T_P.

In the second version, one of the cost metrics is constrained by a hard upper limit. This case can also be modeled with an f function which adds an infinite penalty if the constraint is hurt. A possible interpretation can be the following: if software cost captures execution time, and communication cost captures the extra delay generated by communication, then it makes sense to add them. That is, we define the running time of the system with respect to partition P as $R_P = S_P + C_P$. We suppose that there is a real-time constraint, that is, a constraint on R_P, and the aim is to minimize H_P while satisfying this constraint.

To sum up, the partitioning problems we are dealing with can be formulated as follows:

P1: Given the graph G with the cost functions h, s, and c, and the constants $\alpha, \beta, \gamma \geq 0$, find a hardware/software partition P with minimum T_P.
P2: Given the graph G with the cost functions h, s, and c, and $R_0 \geq 0$, find a hardware/software partition P with $R_P \leq R_0$ that minimizes H_P among all such partitions.

2.2 Extensions to the Basic Model

The basic model of hardware-software partitioning captures many important characteristics of the problem. Its compactness allows us to develop efficient algorithms and helps us better understand the nature of the partitioning problem. However, the basic model can be extended in several ways to incorporate more details.

The problem graph can be extended with dependency information. In this case the communication graph should be rather directed and acyclic. Most authors (see Section 1.1 about previous work) follow this approach.

To respect dependency, the nodes ordered to a context should be scheduled properly; thus the additivity of execution times is not valid anymore. Scheduling requires the execution time of each node; this can be the software/hardware cost or an additional parameter of each node. Whether scheduling is regarded as part of the partitioning or done afterwards is still a question under discussion in the community (recall the different approaches from Section 1.1).

These extensions might be beneficial, but there is a risk that a too complex model can hide the true nature of the problem and only very small instances can be solved. First we show our algorithms for the basic model and then, in Section 4.4, we explain how they can be adapted to these extensions.

3. COMPLEXITY RESULTS

THEOREM 3.1. *The P1 problem can be solved optimally in polynomial time.*

PROOF. We can assume $\alpha = \beta = \gamma = 1$ because otherwise we multiply each h_i by α, each s_i by β, and each c_{ij} by γ. With this modification the problem

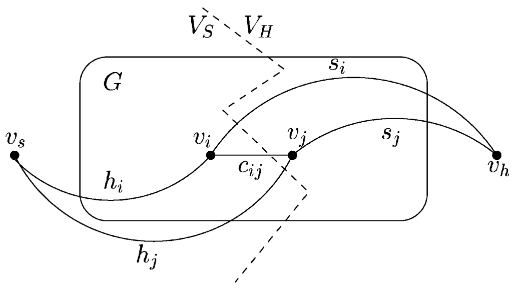


Fig. 1. The auxiliary graph.

becomes similar to the one solved by Stone [1977]. Although Stone handled only one cost metric (time) instead of the linear combination of several cost metrics, the proof of this theorem is identical to the proof by Stone [1977]. The details are omitted; only the main idea of the construction is given to help understand our later algorithms.

We construct an auxiliary graph (see Figure 1) $G' = (V', E')$ based on G as follows: $V' = V \cup \{v_s, v_h\}$, $E' = E \cup E_s \cup E_h$, where $E_s = \{(v, v_s) : v \in V\}$ and $E_h = \{(v, v_h) : v \in V\}$. G' is also a simple, undirected graph, but in G' only the edges are assigned costs; the cost of edge $e \in E'$ is denoted by $b(e)$, and defined as follows:

$$b(e) = \begin{cases} c(e), & \text{if } e \in E, \\ h_i, & \text{if } e = (v_i, v_s) \in E_s, \\ s_i, & \text{if } e = (v_i, v_h) \in E_h. \end{cases}$$

Note that the edges in E_s (i.e., those that connect the vertices to v_s) are assigned the h values, and the edges in E_h are assigned the s values, and not vice versa.

Lemma 3.2 (Stone 1977). *The value of the minimum cut in G' between v_s and v_h is equal to the optimum of the original graph bipartitioning problem.*

By Lemma 3.2, we have reduced the hardware/software partitioning problem to finding a minimum cut between two vertices in a simple undirected graph, for which polynomial-time algorithms are known [Ahuja et al. 1993]. Note that the size of G' is not significantly larger than that of G: if G has n vertices and m edges, then G' has $n+2$ vertices and $m+2n$ edges. This proves the theorem. □

Theorem 3.3. *The P2 problem is $\mathcal{NP}$-hard in the strong sense.*

Proof. The proof can be found in Appendix A. □

These two theorems show that—supposed that $\mathcal{P} \neq \mathcal{NP}$—the P2 problem is significantly harder than the P1 problem. This sheds some light on the origin of complexity in hardware/software partitioning: under the assumption of additivity of costs, the problem is easy if the different cost factors are combined using weighted sum to form a single objective function, whereas it becomes hard if they are bounded or optimized separately.

The other lesson learned from the above two theorems is that not all formulations of the partitioning problem are necessarily $\mathcal{NP}$-hard. The P1 problem, which is apparently easy, is also a meaningful formulation of the hardware/software partitioning problem that can capture a number of real-world variants of the problem. Hence, care has to be taken when claiming that partitioning is $\mathcal{NP}$-hard.

4. ALGORITHMS

4.1 Algorithm for the P1 Problem

The proof of Theorem 3.1 suggests a polynomial-time algorithm for the P1 problem, as summarized in Algorithm 1.

Clearly, the first step of the algorithm can be performed in linear time. For the second step, many algorithms are known. We used the algorithm of Goldberg and Tarjan for finding maximum flow and minimum cut [Goldberg and Tarjan 1988; Cherkassky and Goldberg 1997], which works in $O(n^3)$ time, where n denotes the number of vertices in the graph. Therefore, the whole process can be performed in $O(n^3)$ time.[1] Note that $O(n^3)$ is just a theoretic upper bound for the runtime of Algorithm 1. As will be shown in Section 5, the algorithm is extremely fast in practice.

Note that the condition that α, β, and γ are nonnegative is important because no polynomial-time algorithm is known for finding the minimum cut in a graph with arbitrary edge costs (i.e., where the edge costs are not necessarily nonnegative). In fact, this problem is $\mathcal{NP}$-hard.

It is important to mention that it is not essential that there be exactly three cost metrics to optimize. The same approach works for an arbitrary number of cost metrics as far as the linear combination of them should be minimized. (See Section 4.4 for more details.)

Finally, we note that the algorithm can easily accommodate the following extension to the partitioning model: some components can be fixed to software, while others can be fixed to hardware (e.g., because the other implementation would not make sense or because of some existing components that should be integrated into the system). In this case, the components that are fixed to software are coalesced to form the single vertex v_s, and, similarly, the components that are fixed to hardware are coalesced to form the single vertex v_h. If parallel edges arise, they can be unified to a single edge whose cost is the sum of the costs of the parallel edges. If a loop (i.e., an edge connecting a vertex to itself)

```
Algorithm 1 Polynomial-time algorithm for the P1 problem
1. Create the auxiliary graph.
2. Find a minimum cut in the auxiliary graph.
```

[1]When Stone [1977] published his similar approach, the algorithms for finding a minimum cut in a graph were much slower. In fact, Stone claimed his partitioning algorithm to have $O(n^5)$ running time; thus it was rather impractical.

arises, it can be simply discarded because it does not participate in any cut of the graph.

4.2 Heuristic Algorithm for the P2 Problem

We now show a heuristic for the P2 problem based on Algorithm 1. The idea is to run Algorithm 1 with several different α, β, and γ values. This way, a set of candidate partitions is generated, with the property that each partition is optimal for the P1 problem with some α, β, and γ parameters. Then we select the best partition from this set that fulfills the given limit on R_P.

As already mentioned, the scalability of a heuristic depends on whether the evaluated small fraction of the search space contains high-quality points. We believe that we can achieve this with the above choice of candidate partitions.

Obviously, the result of the run of Algorithm 1 is determined by the *ratio* of the three weights, and not by their absolute values. Therefore, we can fix one of the three, for example, β, and vary only the other two. Thus, we have a two-dimensional search problem, in which the evaluation of a point involves running Algorithm 1 with the appropriate weights.

In order to keep our algorithm fast, we use two phases: in the first phase, we use coarse-grained steps in the two-dimensional plane to find the best valid partitioning approximately, and in the second phase we use a more fine-grained search in the neighborhood of the point found in the first phase (see Algorithm 2 for more details).

In both phases, possible α and γ values are scanned with increments $d\alpha$ and $d\gamma$. Choosing the values for $d\alpha$ and $d\gamma$ constitutes a tradeoff between quality and performance: if small increments are used, then the search is very thorough but slow, if the increments are high, the search becomes fast but superficial. As can be seen in Algorithm 2, we apply a searching scheme that adjusts the increments *dynamically*. More specifically, $d\alpha$ and $d\gamma$ are multiplied with $1+\varepsilon$

Algorithm 2 Heuristic algorithm for the P2 problem.

```
Phase 1: //Scan the whole search space
FOR($\alpha = \alpha_{min}$; $\alpha < \alpha_{max}$; $\alpha = \alpha + d\alpha$)
  FOR($\gamma = \gamma_{min}$; $\gamma < \gamma_{max}$; $\gamma = \gamma + d\gamma$) {
    run Algorithm 1 with parameters $\alpha$, $\beta$, and $\gamma$ to obtain optimal partition $P$;
    IF($R_P \le R_0$ AND $H_P$ < best_so_far) {
      save current solution;
      save previous and next $\alpha$ value ($\alpha_{prev}$, $\alpha_{next}$);
      save previous and next $\gamma$ value ($\gamma_{prev}$, $\gamma_{next}$);
      reset $d\alpha$ and $d\gamma$;
    }
    ELSE
      $d\alpha = (1+\varepsilon)d\alpha$, $d\gamma = (1+\varepsilon)d\gamma$;
  }

Phase 2: //Scan the region around the best point found in Phase 1
reset $d\alpha$ and $d\gamma$;
perform same method as in Phase 1, with $\alpha$ going between $\alpha_{prev}$ and $\alpha_{next}$ and $\gamma$ going
between $\gamma_{prev}$ and $\gamma_{next}$ and using $\varepsilon' < \varepsilon$ instead of $\varepsilon$.
```

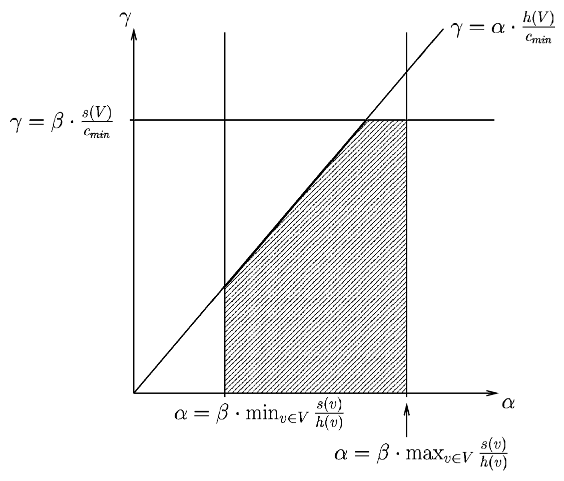

γ
$\gamma = \alpha \cdot \frac{h(V)}{c_{min}}$
$\gamma = \beta \cdot \frac{s(V)}{c_{min}}$
α
$\alpha = \beta \cdot \min_{v \in V} \frac{s(v)}{h(v)}$
$\alpha = \beta \cdot \max_{v \in V} \frac{s(v)}{h(v)}$

Fig. 2. The region to be scanned.

(where ε is a fixed small positive number) in each step when no better solution is found. This way, the algorithm accelerates exponentially in low-quality regions of the search space. On the other hand, $d\alpha$ and $d\gamma$ are reset whenever a better solution is found; thus the search slows down as soon as it finds a better solution. After our initial tests, we fixed $\varepsilon = 0.02$ and $\varepsilon' = 0.01$, which seemed to offer a good tradeoff between speed and quality.

This way, the first phase can find the approximately best values for α and γ, but it is possible that the algorithm jumps over the best values. This is corrected in the second phase. Clearly, this approach works fine if the cost functions are smooth enough and have a relatively simple structure. We will come back to this issue in Section 5.

Finally, it should be mentioned how α_{min}, γ_{min}, α_{max}, and γ_{max} are chosen. The following theorem, the proof of which is omitted for brevity, is useful for this purpose:

THEOREM 4.1. *(i) If $\alpha \leq \beta \cdot \min_{v \in V} \frac{s(v)}{h(v)}$, then the all-hardware partition is optimal with respect to α, β, and γ (regardless of the value of γ).*

(ii) If $\alpha \geq \beta \cdot \max_{v \in V} \frac{s(v)}{h(v)}$, then the all-software partition is optimal with respect to α, β, and γ (regardless of the value of γ).

(iii) Suppose that G is connected, and let c_{min} denote the smallest edge cost. If $\min\left(\alpha \cdot h(V), \beta \cdot s(V)\right) \leq \gamma \cdot c_{min}$, then either the all-hardware or the all-software partition is optimal with respect to α, β, and γ.

Therefore, the region that has to be scanned looks as depicted in Figure 2.

4.3 Determining Lower Bounds

As mentioned earlier, Algorithm 2 can also incorporate the feature of determining lower bounds on the cost of the optimal solution of the given P2 problem instance. This is a unique feature of this algorithm, which is offered by no other competing heuristic. With the help of this feature, Algorithm 2 can maintain an estimate of how far the best solution it found so far is from the optimum. This is very advantageous because it helps evaluate the performance of the algorithm. Moreover, if the lower bound and the found best solution are not far from each other, this may indicate that there is no point in continuing the search. For instance, if the cost values assigned to the nodes and edges of the graph are measured values with a precision of 10%, then there is no point in continuing the search if the gap between the lower bound and the found best solution is under 10% of the lower bound. This way, we can reduce the runtime of the algorithm without any practical loss in the quality of the found solution.

Informally, Algorithm 2 is able to determine the lower bounds because every candidate partition that it evaluates is optimal for the P1 problem with some α, β, and γ values. Hence, each evaluated candidate partition tells us something about the costs of *all* partitions. This is formalized by the following theorem, the proof of which can be found in Appendix B.

THEOREM 4.2. *Suppose that P is an optimal solution of the P1 problem with the weights α, β, and γ. Let Q be any solution of the P2 problem (i.e., a partition that satisfies the bound $R_Q \leq R_0$). Then*

$$H_Q \geq H_P + \frac{\beta S_P + \gamma C_P - \max(\beta, \gamma) R_0}{\alpha}. \tag{1}$$

Note that the right-hand side of (1) contains only known numbers. Therefore, the algorithm can compute a lower bound based on each evaluated candidate partition, and use the best one of these lower bounds. Unfortunately, there is no guarantee that the lower bound will not be far off the optimum. However, as shown in Section 5, the gap between the found best partition and the lower bound was not big for practical benchmarks.

4.4 Adaptation of the Algorithms to Other Partitioning Models

Recall some possible extensions to our basic model from Section 2.2. These extensions can be easily handled by our algorithms without any change in our main idea: try to use the good candidates found by the first algorithm to guide the search of the second algorithm.

The algorithm for P1 can handle any number of cost metrics assigned to software/hardware side, provided their weighted sum should be minimized; for example, every vertex v_i might have a software execution time st_i, a software implementation effort se_i, a hardware execution time ht_i, a hardware chip area ha_i, and further on every cut edge e implies a communication penalty of $c(e)$. If our aim is to minimize $\alpha ST_P + \beta SE_P + \gamma HT_P + \delta HA_P + \varepsilon C_P$ with ST_P, SE_P, HT_P, HA_P defined obviously, then a similar auxiliary graph can be built as in the construction of Theorem 3.1, but the new edge weights $b(e)$ are as follows:

$$b(e) = \begin{cases} \varepsilon c(e), & \text{if } e \in E, \\ \gamma ht_i + \delta ha_i, & \text{if } e = (v_i, v_s) \in E_s, \\ \alpha st_i + \beta se_i, & \text{if } e = (v_i, v_h) \in E_h. \end{cases}$$

Lemma 3.2 will remain true for this graph; hence this extended P1 problem can be similarly solved.

Moreover, scheduling of the tasks can also be incorporated into Algorithm 2. The subroutine of Algorithm 1 returns with a possible solution candidate. One can use any scheduling algorithm available in the literature to evaluate this candidate. The scheduling should be inserted just after the call to Algorithm 1 in line 4 of Algorithm 2. It makes the algorithm more complicated, but it does not change our approach. Therefore, in the test phase we were focusing on the evaluation of the basic algorithms to validate our concept.

5. EMPIRICAL RESULTS

We have implemented the above algorithms using the minimum cut algorithm of Goldberg and Tarjan [Goldberg and Tarjan 1988; Cherkassky and Goldberg 1997]. We had to modify the construction in the proof of Theorem 3.1 slightly because the used minimum cut algorithm works on directed graphs.

Generally, if we want to find the minimum cut in an undirected graph using an algorithm for directed graphs,[2] then we have to change every undirected edge to two directed edges going in opposite directions. However, edges directed to the source or from the sink can be removed, because this does not change the value of the maximum flow, and hence it does not change the value of the minimum cut. In our case, this means that the edges in the original graph are introduced in two copies in the new graph, in opposite directions, but in the case of the additional edges (i.e., edges in E_s and E_h), only one copy is needed, directed to v_h, or from v_s, respectively.

The algorithms have been implemented in C, and tested on a Pentium II 400-MHz PC running SuSE Linux. We conducted two sets of experiments: one for evaluating the performance of Algorithm 1, and one for evaluating the performance and effectiveness of Algorithm 2.

5.1 Experience with Algorithm 1

Since Algorithm 1 finds the optimal solution for the partitioning problem at hand, we only had to test its speed on practical problem instances. (Recall from Section 4.1 that it is an $O(n^3)$ algorithm, but this is only an asymptotic upper bound on its running time.)

For testing Algorithm 1, several random graphs of different sizes and with random costs have been used. In order to reduce the number of test runs and the amount of test data to process, we fixed the ratio of edges and vertices in the test graphs to 2, which means that, on average, each vertex has four neighbors. Previous experience with real-world task graphs [Arató et al. 2003a] has shown that this average is typical.

Table I shows measurement results concerning the running time of Algorithm 1 on graphs of different sizes. As can be seen, the algorithm is extremely fast: it finds the optimum in the case of a graph with 10,000 vertices and 20,000

[2]There are also algorithms for finding the minimum cut in an undirected graph, which are even faster than the ones for directed graphs. However, we need a cut that separates two given vertices (a so-called st-cut), and, for this problem, no faster algorithms are known for the undirected case.

Table I. Running Time of Algorithm 1

n	Running time of Algorithm 1 (s)
100	0.0007
1000	0.0198
3000	0.0666
5000	0.1264
7000	0.1965
10000	0.2896

edges in less than 0.3 s. Moreover, the practical running time of the algorithm seems to be roughly linear.

5.2 Experience with Algorithm 2

Since Algorithm 2 is a heuristic rather than an exact algorithm, we had to determine both its performance and the quality of the solutions it finds, empirically. For this purpose, we compared it with two other heuristics that are widely used for hardware/software partitioning: a genetic algorithm (GA) and an improved Kernighan/Lin–type heuristic (KL). In the case of the GA it is important to tune its parameters to match the characteristics of the problem domain. Details on this can be found in Arató et al. [2003a].

In the case of the KL algorithm, we built on the improvements suggested by Vahid and Le [1997]. Specifically, Vahid and Le [1997] defined the following changes: (i) they redefined a move as a single node move, rather than a swap; (ii) they described an efficient data structure; (iii) they replaced the cut metric of the original KL heuristic by a more complex metric. We made use of these changes, the only difference being that our cost function was slightly different from theirs:

$$cost(P) = \begin{cases} \infty, & \text{if } R_P > R_0, \\ H_P, & \text{otherwise.} \end{cases}$$

Note that Vahid and Le [1997] used the DAG property of their graph representation to show that a move has only local effect, which is important for the performance of the algorithm. This was also true in our case: by moving a node from hardware to software or vice versa, only the gain value of its neighbors can change.

For testing, we used benchmarks from MiBench [Guthaus et al. 1997] and our own designs, as well as bigger, random graphs. The characteristics of the test cases are summarized in Table II. n and m denote the number of nodes and edges, respectively, in the communication graph. *Size* denotes the length of the description of the graph (the performance of an algorithm is usually evaluated as a function of the length of the input). We calculated the size as $2n + 3m$ because each node is assigned two values—its hardware and software costs—and each edge is assigned three numbers—the IDs of its endpoints and its communication cost.

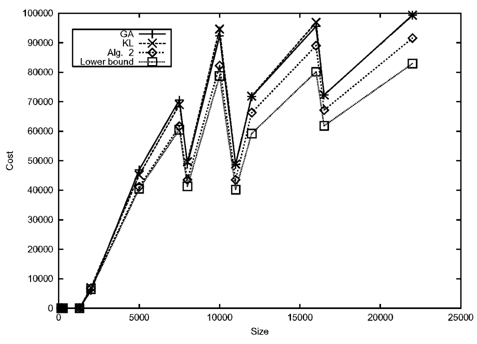
Where software costs were not available, they were generated as uniform random numbers from the interval [1, 100]. Where hardware costs were not

Table II. Summary of the Used Benchmarks

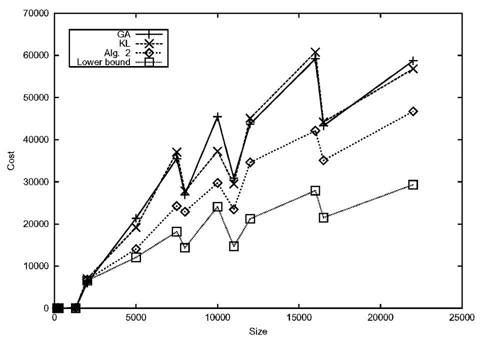
Name	n	m	Size	Description
crc32	25	34	152	32-bit cyclic redundancy check. From the Telecommunications category of MiBench.
patricia	21	50	192	Routine to insert values into Patricia tries, which are used to store routing tables. From the Network category of MiBench.
dijkstra	26	71	265	Computes shortest paths in a graph. From the Network category of MiBench.
clustering	150	333	1299	Image segmentation algorithm in a medical application.
rc6	329	448	2002	RC6 cryptographic algorithm.
random1	1000	1000	5000	Random graph.
random2	1000	2000	8000	Random graph.
random3	1000	3000	11000	Random graph.
random4	1500	1500	7500	Random graph.
random5	1500	3000	12000	Random graph.
random6	1500	4500	16500	Random graph.
random7	2000	2000	10000	Random graph.
random8	2000	4000	16000	Random graph.
random9	2000	6000	22000	Random graph.

available, they were generated as random numbers from a normal distribution with expected value $\kappa \cdot s_i$ and standard deviation $\lambda \cdot \kappa \cdot s_i$, where s_i is the software cost of the given node. That is, there is a correlation, as defined by the value of λ, between a node's hardware and software costs. This corresponds to the fact that more complicated components tend to have both higher software and higher hardware costs. We tested two different values for λ: 0.1 (high correlation) and 0.6 (low correlation). The value of κ only corresponds to the choice of units for software and hardware costs, and thus it has no algorithmic implications. The communication costs were generated as uniform random numbers from the interval $[0, 2 \cdot \mu \cdot s_{max}]$, where s_{max} is the highest software cost. Thus, communication costs have an expected value of $\mu \cdot s_{max}$, and μ is the so-called communication to computation ratio (CCR). We tested two different values for μ: 1 (computation-intensive case) and 10 (communication-intensive case). Finally, the limit R_0 was taken from the interval $[0, \sum s_i]$. Note that $R_0 = 0$ means that all components have to be mapped to hardware, whereas $R_0 = \sum s_i$ means that all components can be mapped to software. All sensible values of R_0 lie between these two extremes. We tested two values for R_0: one generated as a uniform random number from the interval $[0, \frac{1}{2} \sum s_i]$ (strict real-time constraint) and one taken randomly from $[\frac{1}{2} \sum s_i, \sum s_i]$ (loose real-time constraint).

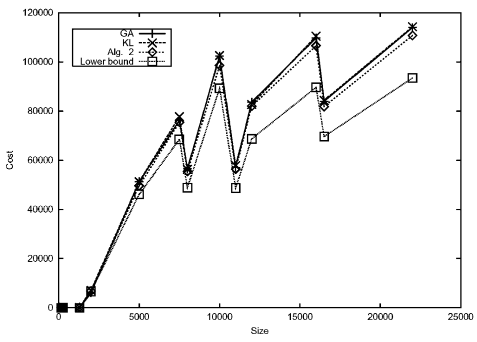
So we tested the three algorithms on the above set of problems, using two values for each of the three parameters (correlation between hardware and software costs, CCR, R_0). However, we found that the correlation between hardware and software costs did not have any significant impact on the performance of the algorithms. Therefore we include four plots, according to the combinations of the two remaining parameters, on the quality of the solutions found by the algorithms (see Figure 3). Since the objective was to minimize costs, smaller values are better. The lower bounds produced by Algorithm 2



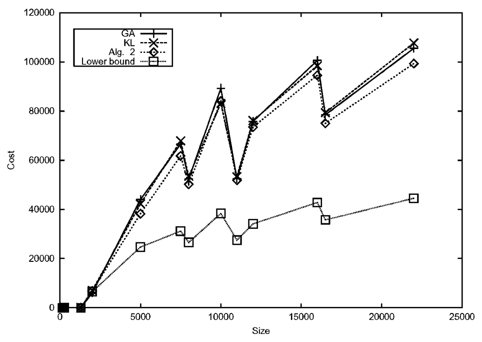
(a) CCR=1, R_0=low



(b) CCR=1, R_0=high



(c) CCR=10, R_0=low



(d) CCR=10, R_0=high

Fig. 3. Algorithm 2 versus GA and KL: quality of found solution.

are also shown. Based on the diagrams, the following observations can be made:

—For relatively small graphs, all three heuristics yield equal or very similar results, regardless of the parameter settings. Moreover, these results are very close to the lower bound computed by Algorithm 2, meaning that they are at least near-optimal.

—For bigger graphs, Algorithm 2 consistently outperforms the other two heuristics. This is especially true in the low-CCR cases. In the high-CCR cases, the difference between the algorithms is not so striking. This is probably due to the easier nature of these problem instances (note that with growing CCR, the partitioning problem becomes essentially a simple minimum cut problem with polynomial complexity). Moreover, the difference between the results of Algorithm 2 and the other two heuristics is clearly growing.

—The results found by GA and KL are very similar, but in most cases the GA is slightly better.

—The results found by Algorithm 2 are in most cases not very far from the lower bounds it produced—the difference was 31% on average. Of course the difference keeps growing with bigger graphs, but quite slowly. This proves the high quality of both the solutions and the lower bounds found by our algorithm.

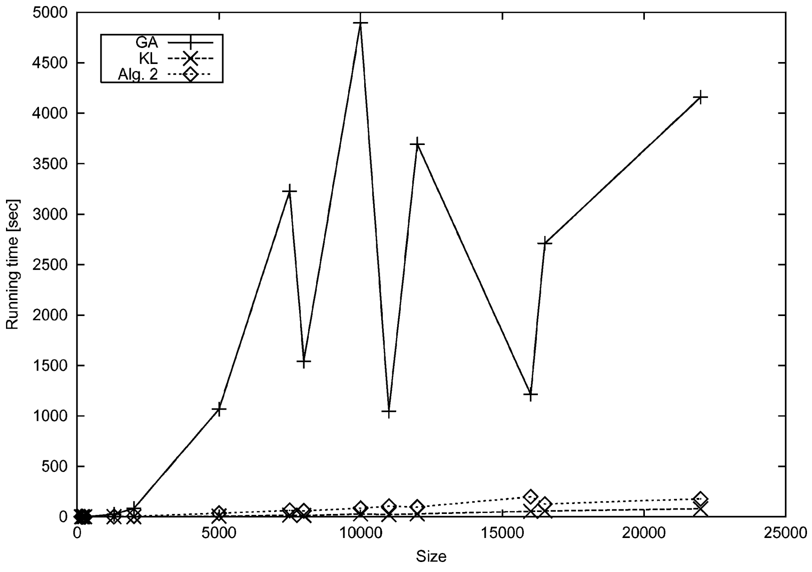


Fig. 4. Algorithm 2 versus GA and KL: running time.

—The choice of the R_0 parameter does not seem to significantly impact the relative performance of the algorithms. However, the lower bounds produced by Algorithm 2 do seem to be sensitive to this parameter: they are clearly better for low R_0 values.

An even bigger difference between the three algorithms is their running time, which is shown in Figure 4 (here, only one plot is shown because the explained parameter settings do not have a significant impact on the running times). We can observe the following:

—Again, for relatively small graphs, the speed of the algorithms was comparable. However, for bigger graphs, KL and Algorithm 2 were much faster than GA, and again, the difference kept growing with bigger graphs. For the biggest graphs, GA was about 20 times slower than Algorithm 2.

—The running time of GA oscillated wildly. In some cases, it took over an hour for the GA to terminate. However, even the shortest GA runs were much slower than the other two algorithms.

—The fastest of the three was clearly the KL algorithm: for small graphs, it was about five times faster than Algorithm 2, but for bigger graphs the difference decreased. For the biggest graphs, KL was about 2.5 times faster than Algorithm 2.

—The speed of both KL and Algorithm 2 is acceptable because both could solve even the biggest problems in 2–3 min, and the smaller ones in a couple of seconds.

Another question that we addressed empirically was whether or not the two-dimensional search approach of Algorithm 2 is adequate.

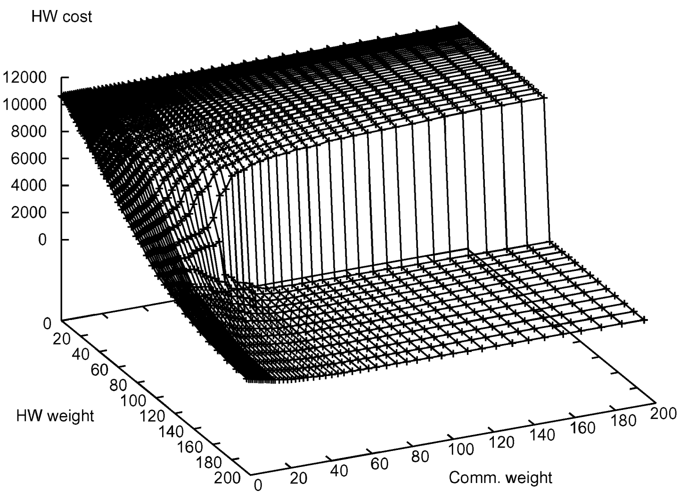


Fig. 5. Hardware cost of the optimal partition in the P1 problem as the function of the weights α and γ.

The above results show that the algorithm performed very well on large benchmarks. This can be attributed to the smoothness of the costs as functions of the weights α, β, and γ. An example can be seen in Figure 5 showing the hardware cost of the optimal partition in the P1 problem for different values of the hardware weight α and communication weight γ (the software weight β was fixed to 100). Notice the smoothness and the simple structure of this function. Actually, it can be proven that this function is monotonously decreasing in α.

In some test cases, we also ran a modified version of Algorithm 2 in which the two-dimensional search space was searched uniformly in small steps, without augmenting $d\alpha$ and $d\gamma$. The test results showed no improvement in the results; however, the speed of the algorithm worsened significantly. This justifies the search strategy of Algorithm 2.

To sum up: Algorithm 2 offers a clear advantage over the other two heuristics concerning the quality of the found solution. It is at the same time significantly faster than GA, and somewhat slower than KL, but still fast enough to be applicable in practice. Moreover, it produces high-quality lower bounds.

6. CONCLUSION

In this article, we defined two slightly different versions of the hardware/software partitioning problem (P1 and P2). We proved that the P1 problem can be solved in polynomial time, but the P2 problem is $\mathcal{NP}$-hard. The polynomial-time algorithm for the P1 problem (Algorithm 1) makes use of the graph-theoretic properties of the hardware/software partitioning problem. It has a worst-case running time of $O(n^3)$ steps, but our empirical experiments showed that on practical examples it is very fast.

Based on this algorithm, we also proposed a new heuristic for the P2 problem (Algorithm 2) which works by running Algorithm 1 with several different weights to obtain high-quality candidate partitions, from which it chooses the best one satisfying the given constraint.

Algorithm 2 possesses the unique feature that it can calculate lower bounds for the optimal solution and hence it can evaluate how far its currently found best solution lies from the optimum.

In our empirical tests on several benchmarks we compared Algorithm 2 with two established partitioners: a genetic algorithm and an improved Kernighan/Lin–type algorithm. We found that our algorithm consistently outperformed the other two heuristics, while being slightly slower than KL and significantly faster than GA. We attribute the good scalability of our algorithm to the fact that it only evaluates high-quality points of the search space (only those that are optimal solutions of the P1 problem for some weights) and hence it makes better use of the combinatorial properties of the search problem.

Generalization of our algorithms for multiway partitioning and proving or disproving approximation bounds for Algorithm 2 remain interesting future research directions.

APPENDIX

A. $\mathcal{NP}$–Hardness Results

THEOREM A.1. *The P2 problem is $\mathcal{NP}$-hard.*

PROOF. The proof can be found in the paper of Mann and Orbán [2003]. □

However, the P2 problem is $\mathcal{NP}$–hard in the *strong sense* as well, that is, even if the vertex and edge costs have to be polynomial in n. In the following we show a reduction of the minimum bisection problem to P2.

THEOREM A.2. *The P2 problem is $\mathcal{NP}$-hard in the strong sense.*

PROOF. We reduce the decision version of the minimum bisection problem, which is known to be $\mathcal{NP}$-complete [Garey and Johnson 1979], to P2.

Given an instance of the minimum bisection problem on $G(V, E)$ with n vertices, where n is even, m edges, and a limit K, our goal is to find a cut (A, B), for which $|A| = |B| = \frac{n}{2}$ and the cutsize is at most K $(K \le m)$.

Now associate the following instance of the P2 problem to it. Let $h(v_i) = s(v_i) = 1$ for each $v_i \in V$ and let $c(v_i, v_j) = \frac{1}{m+1}$ for each $(i, j) \in E$. Define $R_0 := \frac{n}{2} + \frac{K}{m+1}$. Clearly this instance has polynomial costs in n.

For $X, Y \subseteq V$ we denote by $m(X, Y)$ the number of edges between X and Y and by $c(X, Y)$ the total cost of edges between X and Y.

We claim that there exists a feasible bisection iff the optimum for the P2 problem is at most $\frac{n}{2}$. Indeed, if (A, B) is a solution for the bisection problem ($|A| = |B| = \frac{n}{2}$ and $m(A, B) \le K$), then (A, B) is also a feasible solution for P2, since $s(B) + c(A, B) = |B| + \frac{1}{m+1} m(A, B) \le \frac{n}{2} + \frac{K}{m+1} = R_0$. The hardware cost of (A, B) in P2 is $h(A) = |A| = \frac{n}{2}$, and thus the optimum is at most $\frac{n}{2}$.

Vice versa, if in the optimal partition (V_H, V_S) of P2 the hardware cost is at most $\frac{n}{2}$, then $h(V_H) = |V_H| \le \frac{n}{2}$ and $s(V_S) + c(V_H, V_S) \le \frac{n}{2} + \frac{K}{m+1} < \frac{n}{2} + 1$; thus $s(V_S) = |V_S| \le \frac{n}{2}$, as it is an integer and c is nonnegative. As both sides of the partition (V_H, V_S) are not larger than $\frac{n}{2}$, $|V_H| = |V_S| = \frac{n}{2}$ must hold. This also implies—using again the condition for the running time—that $c(V_H, V_S) \leq \frac{K}{m+1}$, and hence $m(V_H, V_S) \leq K$. So (V_H, V_S) is indeed a solution for the bisection problem as well. □

B. Lower Bound

THEOREM B.1. *Suppose that P is an optimal solution of the P1 problem with the weights α, β, and γ. Let Q be any solution of the P2 problem (i.e., a partition that satisfies the bound $R_Q \leq R_0$). Then*

$$H_Q \geq H_P + \frac{\beta S_P + \gamma C_P - \max(\beta, \gamma) R_0}{\alpha}. \tag{2}$$

PROOF. Since P is optimal with respect to the weights α, β, and γ, it follows that

$$\alpha H_P + \beta S_P + \gamma C_P \leq \alpha H_Q + \beta S_Q + \gamma C_Q$$

and hence

$$H_Q \geq H_P + \frac{\beta S_P + \gamma C_P - \beta S_Q - \gamma C_Q}{\alpha}. \tag{3}$$

Of course, this is also a lower bound on H_Q, but the right-hand side cannot be computed because S_Q and C_Q are not known. However, since Q is a valid partition, it follows that

$$S_Q + C_Q = R_Q \leq R_0$$

and therefore

$$\beta S_Q + \gamma C_Q \leq \max(\beta, \gamma) S_Q + \max(\beta, \gamma) C_Q = \max(\beta, \gamma) R_Q \leq \max(\beta, \gamma) R_0.$$

Substituting this into (3) proves the theorem. □

REFERENCES

ABDELZAHER, T. F. AND SHIN, K. G. 2000. Period-based load partitioning and assignment for large real-time applications. *IEEE Trans. Comput. 49*, 1, 81–87.

AHUJA, R. K., MAGNANTI, T. L., AND ORLIN, J. B. 1993. *Network Flows: Theory, Algorithms, and Applications*. Prentice-Hall, Englewood Cliffs, NJ.

ARATÓ, P., JUHÁSZ, S., MANN, Z. Á., ORBÁN, A., AND PAPP, D. 2003a. Hardware/software partitioning in embedded system design. In *Proceedings of the IEEE International Symposium on Intelligent Signal Processing*.

ARATÓ, P., MANN, Z. Á., AND ORBÁN, A. 2003b. Hardware-software co-design for Kohonen's self-organizing map. In *Proceedings of the IEEE 7th International Conference on Intelligent Engineering Systems*.

BARROS, E., ROSENSTIEL, W., AND XIONG, X. 1993. Hardware/software partitioning with UNITY. In *Proceedings of the 2nd International Workshop on Hardware-Software Codesign*.

BINH, N. N., IMAI, M., SHIOMI, A., AND HIKICHI, N. 1996. A hardware/software partitioning algorithm for designing pipelined ASIPs with least gate counts. In *Proceedings of the 33rd Design Automation Conference*.

CHATHA, K. S. AND VEMURI, R. 2001. MAGELLAN: Multiway hardware-software partitioning and scheduling for latency minimization of hierarchical control-dataflow task graphs. In *Proceedings of CODES 01*.

CHERKASSKY, B. V. AND GOLDBERG, A. V. 1997. On implementing push-relabel method for the maximum flow problem. *Algorithmica 19*, 4, 390–410.

DASDAN, A. AND AYKANAT, C. 1997. Two novel multiway circuit partitioning algorithms using relaxed locking. *IEEE Trans. Comput.-Aid. Des. Integr. Circ. Syst. 16*, 2 (Feb.), 169–177.

DICK, R. P. AND JHA, N. K. 1998. MOGAC: A multiobjective genetic algorithm for hardware-software co-synthesis of hierarchical heterogeneous distributed embedded systems. *IEEE Trans. Comput.-Aid. Des. Integr. Circ. Syst. 17*, 10, 920–935.

ELES, P., PENG, Z., KUCHCINSKI, K., AND DOBOLI, A. 1996. Hardware/software partitioning of VHDL system specifications. In *Proceedings of EURO-DAC '96*.

ELES, P., PENG, Z., KUCHCINSKI, K., AND DOBOLI, A. 1997. System level hardware/software partitioning based on simulated annealing and tabu search. *Des. Automat. Embedd. Syst. 2*, 1 (Jan.), 5–32.

ERNST, R., HENKEL, J., AND BENNER, T. 1993. Hardware/software cosynthesis for microcontrollers. *IEEE Des. Test Comp. 10*, 4, 64–75.

FIDUCCIA, C. M. AND MATTHEYSES, R. M. 1982. A linear-time heuristic for improving network partitions. In *Proceedings of the 19th Design Automation Conference*.

GAREY, M. R. AND JOHNSON, D. S. 1979. *A Guide to the Theory of NP–Completeness*. Freeman, San Francisco, CA.

GOLDBERG, A. V. AND TARJAN, R. E. 1988. A new approach to the maximum flow problem. *J. Assoc. Comput. Mach. 35*, 921–940.

GRODE, J., KNUDSEN, P. V., AND MADSEN, J. 1998. Hardware resource allocation for hardware/software partitioning in the LYCOS system. In *Proceedings of Design Automation and Test in Europe (DATE '98)*.

GUPTA, R. K. AND DE MICHELI, G. 1993. Hardware-software cosynthesis for digital systems. *IEEE Des. Test Comp. 10*, 3, 29–41.

GUTHAUS, M. R., RINGENBERG, J. S., ERNST, D., AUSTIN, T. M., MUDGE, T., AND BROWN, R. B. 1997. MiBench: A free, commercially representative embedded benchmark suite. In *Proceedings of the IEEE 4th Annual Workshop on Workload Characterization*.

HENKEL, J. AND ERNST, R. 2001. An approach to automated hardware/software partitioning using a flexible granularity that is driven by high-level estimation techniques. *IEEE Trans. VLSI Syst. 9*, 2, 273–289.

KALAVADE, A. 1995. System-level codesign of mixed hardware-software systems. Ph.D. dissertation. University of California, Berkeley, Berkeley, CA.

KALAVADE, A. AND LEE, E. A. 1997. The extended partitioning problem: Hardware/Software mapping, scheduling and implementation-bin selection. *Des. Automat. Embedd. Syst. 2*, 2, 125–164.

KALAVADE, A. AND SUBRAHMANYAM, P. A. 1998. Hardware/software partitioning for multifunction systems. *IEEE Trans. Comput.-Aid. Des. Integr. Circ. Syst. 17*, 9 (Sept.), 819–837.

KERNIGHAN, B. W. AND LIN, S. 1970. An efficient heuristic procedure for partitioning graphs. *Bell Syst. Techn. J. 49*, 2, 291–307.

LOPEZ-VALLEJO, M., GRAJAL, J., AND LOPEZ, J. C. 2000. Constraint-driven system partitioning. In *Proceedings of DATE*. 411–416.

LOPEZ-VALLEJO, M. AND LOPEZ, J. C. 1998. A knowledge based system for hardware-software partitioning. In *Proceedings of DATE*.

LOPEZ-VALLEJO, M. AND LOPEZ, J. C. 2003. On the hardware-software partitioning problem: System modeling and partitioning techniques. *ACM Trans. Des. Automat. Electron. Syst. 8*, 3 (July), 269–297.

MADSEN, J., GRODE, J., KNUDSEN, P. V., PETERSEN, M. E., AND HAXTHAUSEN, A. 1997. LYCOS: The Lyngby co-synthesis system. *Des. Automat. Embedd. Syst. 2*, 2, 195–236.

MANN, Z. Á. AND ORBÁN, A. 2003. Optimization problems in system-level synthesis. In *Proceedings of the 3rd Hungarian-Japanese Symposium on Discrete Mathematics and Its Applications*.

MEI, B., SCHAUMONT, P., AND VERNALDE, S. 2000. A hardware/software partitioning and scheduling algorithm for dynamically reconfigurable embedded systems. In *Proceedings of ProRISC*.

NIEMANN, R. 1998. *Hardware/Software Co-Design for Data Flow Dominated Embedded Systems*. Kluwer Academic Publishers, Norwell, MA.

NIEMANN, R. AND MARWEDEL, P. 1997. An algorithm for hardware/software partitioning using mixed integer linear programming. *Des. Automat. Embedd. Syst.* Special Issue: Partitioning Methods for Embedded Systems *2*, 165–193.

O'NILS, M., JANTSCH, A., HEMANI, A., AND TENHUNEN, H. 1995. Interactive hardware-software partitioning and memory allocation based on data transfer profiling. In *Proceedings of the International Conference on Recent Advances in Mechatronics*.

QIN, S. AND HE, J. 2000. An algebraic approach to hardware/software partitioning. Tech. rep. 206, UNU/IIST.

QUAN, G., HU, X., AND GREENWOOD, G. 1999. Preference-driven hierarchical hardware/software partitioning. In *Proceedings of the IEEE/ACM International Conference on Computer Design*.

SAAB, Y. G. 1995. A fast and robust network bisection algorithm. *IEEE Trans. Comput. 44*, 7 (July), 903–913.

SRINIVASAN, V., RADHAKRISHNAN, S., AND VEMURI, R. 1998. Hardware software partitioning with integrated hardware design space exploration. In *Proceedings of DATE*.

STITT, G., LYSECKY, R., AND VAHID, F. 2003. Dynamic hardware/software partitioning: A first approach. In *Proceedings of DAC*.

STONE, H. 1977. Multiprocessor scheduling with the aid of network flow algorithms. *IEEE Trans. Softw. Eng. 3*, 1 (Jan.), 85–93.

VAHID, F. 1997. Modifying min-cut for hardware and software functional partitioning. In *Proceedings of the International Workshop on Hardware-Software Codesign*.

VAHID, F. 2002. Partitioning sequential programs for CAD using a three-step approach. *ACM Trans. Des. Automat. Electron. Syst. 7*, 3 (July), 413–429.

VAHID, F. AND GAJSKI, D. 1995. Clustering for improved system-level functional partitioning. In *Proceedings of the 8th International Symposium on System Synthesis*.

VAHID, F. AND LE, T. D. 1997. Extending the Kernighan/Lin heuristic for hardware and software functional partitioning. *Des. Automat. Embedd. Syst. 2*, 237–261.

WOLF, W. H. 1997. An architectural co-synthesis algorithm for distributed embedded computing systems. *IEEE Trans. VLSI Syst. 5*, 2 (June), 218–229.

Received January 2004; revised May 2004, August 2004; accepted August 2004

A Unified Method for Phase Shifter Computation

DIMITRI KAGARIS
Southern Illinois University

Phase shifters are used to shift the bit sequences produced by the successive stages of a built-in test pattern generator (TPG) based on a linear finite state machine (LFSM) by a specified amount (*phase shift*) relative to the characteristic sequence. An upper bound on the number of taps to be used for each phase shifter and a lower bound on the phase-shift value between successive stages of the TPG mechanism are the general parameters of the problem. Methods to design such phase shifters have been given in the past separately for Type-1 LFSRs, Type-2 LFSRs, and three-neighborhood cellular automata. In this article, we show how phase shifters can be synthesized uniformly and efficiently for any LFSM, including the aforementioned ones. We demonstrate the method by showing how to obtain phase shifters for two-dimensional cellular automata and for ring generators.

Categories and Subject Descriptors: B.8.1 [**Performance and Reliability**]: Reliability, Testing, and Fault Tolerance

General Terms: Algorithms, Reliability, Theory

Additional Key Words and Phrases: Test pattern generation (TPG), built-in self-test (BIST), phase shifters, linear finite state machines, linear feedback shift registers, cellular automata

1. INTRODUCTION

The bit sequences generated by successive cells of built-in test pattern generators (TPGs) suffer in general from correlations and/or linear dependencies. This is problematic for pseudorandom and/or pseudoexhaustive generation of test patterns [Bardell et al. 1987]. Of the three popular built-in TPGs, namely, Type-1 (external XOR) linear feedback shift registers (LFSRs), Type-2 (internal XOR) LFSRs, and three-neighborhood cellular automata (CA), the problem is more apparent for Type-1 LFSRs since the bit sequences produced by successive LFSR stages are simply shifted versions by 1 bit of each other. Nevertheless, as experimental study has shown [Mrugalski et al. 1999], the problem persists in the other classes as well. To rectify this behavior, and, more generally, to offer

Author's address: Department of Electrical and Computer Engineering, Southern Illinois University, Carbondale, IL 62901-6603; email: kagaris@engr.siu.edu.

the designer freedom in choosing the relative phase shifts of the stages and not be bound by the inherent phase shifts of the stages of the TPG mechanism in use, networks of XOR gates, known as *phase shifters*, have been proposed in the literature [Bardell et al. 1987; Mrugalski et al. 2000; Rajski et al. 1998] to be inserted between the LFSR or CA cells and the test inputs (primary inputs, test-phase inputs, and/or scan chain inputs) of the circuit under test. A phase shifter is basically a multi-input XOR gate driven by an appropriate subset of stages of the TPG mechanism in use. It is known that the bit sequence produced by any stage of a linear finite state machine- (LFSM-) based TPG is a shifted version of the characteristic sequence of that TPG. Following the shift-and-add property, the bit sequence produced at the output of a phase shifter is again a shifted version of the characteristic sequence. The interest is that the number of shift positions (*phase shifts*) by which the bit sequence of one stage differs from that of another is fully controllable and can be made to be any number k, $2 \leq k \leq p$, where p is the length of the characteristic sequence (or equivalently, the period of the characteristic polynomial). This can be done by determining an appropriate subset of TPG mechanism stages (*taps*) to serve as inputs to the phase shifter XOR gate. The usual requirements in the phase shifter design are an upper bound B on the number of taps per phase shifter (to control the hardware overhead) and a lower bound L on the attained phase shift between successive TPG stages. Methods to obtain the appropriate subset of stages for this purpose have been given in Mrugalski et al. [2000] and Rajski et al. [1998], provided that the TPG in use is a Type-1 LFSR, a Type-2 LFSR, or a CA (a different algorithm is given separately for each case). In this article, we show a new, unified method that applies to *any* kind of LFSM. The method is as efficient and straightforward to implement as the special case methods.

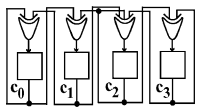
The rest of the article is organized as follows: Section 2 gives a short overview of the existing methods for the cases of Type-1 LFSR, Type-2 LFSR, and CA. Section 3 describes the new unified method. Section 4 presents two case studies on the application of the method to the computation of phase shifters for two nontraditional TPG mechanisms, namely, two-dimensional CA and ring generators. Section 5 concludes the article.

2. OVERVIEW OF EXISTING METHODS

Assume that the TPG mechanism in use is a three-neighborhood CA. An example is given in Figure 1. Given a requested phase shift k for a particular stage with respect to a reference sequence, an effective technique to find which cells of the CA should be connected into an XOR gate (phase shifter) so that the resulting bit sequence is a shifted version by k positions of the reference sequence was given in Mrugalski et al. [2000] based on a previous result by Ireland and Marshall [1968]. If the reference sequence is cell j of the CA, and one wants a sequence that has a phase shift of k with respect to that, then do the following: (i) initialize the CA with a vector that contains only one 1 at position j; (ii) simulate the CA for k cycles; (iii) read off the produced vector: the positions of the 1s in it indicate the cells that should be XORed together to yield the requested sequence. Given an upper bound B on the number of taps per

CA

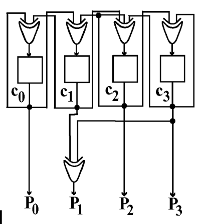
$P(x) = x^4 + x^3 + 1$
Rule: [1 0 1 1]



Cyc	Stages 0 1 2 3	
0	1 0 0 0	← 0
1	1 1 0 0	
2	0 1 1 0	
3	1 1 0 1	
4	0 1 0 1	← 4
5	1 0 0 1	
6	1 1 1 1	
7	0 0 1 0	← 7
8	0 1 1 1	
9	1 1 1 0	
10	0 0 0 1	← 10
11	0 0 1 1	
12	0 1 0 0	
13	1 0 1 0	
14	1 0 1 1	

(a)

CA + Phase Shifters
[0,4,7,10]



$$H = \begin{bmatrix} 1\,0\,0\,0 \\ 0\,1\,0\,1 \\ 0\,0\,1\,0 \\ 0\,0\,0\,1 \end{bmatrix}$$

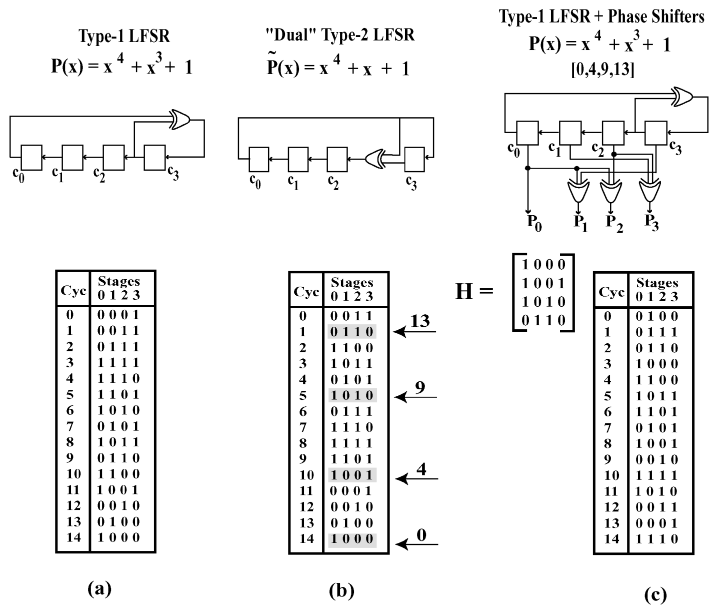
Cyc	Stages 0 1 2 3
0	1 0 0 0
1	1 1 0 0
2	0 1 1 0
3	1 0 0 1
4	0 0 0 1
5	1 1 0 1
6	1 0 1 1
7	0 0 1 0
8	0 0 1 1
9	1 1 1 0
10	0 1 0 1
11	0 1 1 1
12	0 1 0 0
13	1 0 1 0
14	1 1 1 1

(b)

Fig. 1. Obtaining phase shifts for $L = 3$ and $B = 2$ for a CA with rule vector [1011] (characteristic polynomial $P(x) = x^4 + x^3 + 1$). (a) Original CA and state sequence starting with [1000]. (b) CA with the required phase shifters and resulting output sequence.

phase shifter and a lower bound L on the phase shift between successive stages, the above simulation procedure is used as follows: simulate the CA for at least L more cycles after the last satisfying vector found. If the current vector has at most B 1s in it, then use it as the next satisfying vector; otherwise continue simulating the CA. Techniques to better organize this basic search framework have also been given in Mrugalski et al. [2000]. For example, vectors containing at most B 1s can be generated lexicographically and used as initial states in simulations of L cycles, checking each time if there is a conflict with the ranges of previously discovered phase-shift vectors satisfying the requirements.

As an illustration, assume that for the CA of Figure 1(a) we want to obtain successive phase shifts that are at least $L = 3$ apart and can be attained with at most $B = 2$ taps per phase shifter. Assuming that the sequence of stage 0 is the reference sequence, we see that we can use the vectors at cycles 4, 7, and 10, that is, vectors [0101], [0010], and [0001]. These vectors along with [1000], which serves to indicate the reference sequence, are shown as rows of *phase-shift matrix H*. The specific values of the phase shifts attained are indicated by the notation [0, 4, 7, 10]. The corresponding CA and the phase shift logic is shown in Figure 1(b), along with the resulting output sequence. As can be verified, the bit sequence at each output stage i, $0 \leq i \leq 3$, has the appropriate phase shift with respect to the sequence of stage 0, and is the XOR of the bit sequences of those stages in Figure 1(a) that correspond to the 1s in the ith



Cyc	Stages 0 1 2 3
0	0 0 0 1
1	0 0 1 1
2	0 1 1 1
3	1 1 1 1
4	1 1 1 0
5	1 1 0 1
6	1 0 1 0
7	0 1 0 1
8	1 0 1 1
9	0 1 1 0
10	1 1 0 0
11	1 0 0 1
12	0 0 1 0
13	0 1 0 0
14	1 0 0 0

Cyc	Stages 0 1 2 3
0	0 0 1 1
1	0 1 1 0
2	1 1 0 0
3	1 0 1 1
4	0 1 0 1
5	1 0 1 0
6	0 1 1 1
7	1 1 1 0
8	1 1 1 1
9	1 1 0 1
10	1 0 0 1
11	0 0 0 1
12	0 0 1 0
13	0 1 0 0
14	1 0 0 0

$$H = \begin{bmatrix} 1&0&0&0 \\ 1&0&0&1 \\ 1&0&1&0 \\ 0&1&1&0 \end{bmatrix}$$

Cyc	Stages 0 1 2 3
0	0 1 0 0
1	0 1 1 1
2	0 1 1 0
3	1 0 0 0
4	1 1 0 0
5	1 0 1 1
6	1 1 0 1
7	0 1 0 1
8	1 0 0 1
9	0 0 1 0
10	1 1 1 1
11	1 0 1 0
12	0 0 1 1
13	0 0 0 1
14	1 1 1 0

Fig. 2. Obtaining phase shifts for $L = 4$ and $B = 2$ for a Type-1 LFSR (characteristic polynomial $x^4 + x^3 + 1$). (a) Original Type-1 LFSR and state sequence. (b) Dual Type-2 LFSR and state sequence ending with vector [1000]. (c) Type-1 LFSR with the required phase shifters and resulting output sequence.

row of H. (In this article, we use the convention that the rows of a matrix are indexed starting with 0.)

For the case of a Type-1 or Type-2 LFSRs, the procedure is as follows [Rajski et al. 1998]: (i) for a Type-1 (Type-2) LFSR with characteristic polynomial $P(x)$, consider the "dual" Type-2 (Type-1) LFSR with characteristic polynomial the reciprocal of $P(x)$, that is, $\tilde{P}(x) = x^d P(\frac{1}{x})$; (ii) initialize the dual LFSR with a vector that contains only one 1 at position j; (iii) simulate the dual LFSR for k cycles, going in the *reverse* direction; (iii) read off the produced vector: the positions of the 1s in it indicate the stages of the original LFSR that should be XORed together to yield the requested sequence. Similarly as before, this simulation procedure provides the basis for the search procedure for the bounded tap case.

As an illustration, assume that for the Type-1 LFSR of Figure 2(a) we want to obtain successive phase shifts that are at least $L = 4$ apart and can be attained with at most $B = 2$ taps per phase shifter. Then, following the technique of Rajski et al. [1998], we consider the "dual" Type-2 LFSR with the reciprocal characteristic polynomial (Figure 2(b)) and we start from vector 1000, going in the reverse direction. The vectors at cycles 10, 5, and 1, that is, [1001], [1010], and [0110], satisfy the requirements. The corresponding phase-shift matrix H and the original Type-1 LFSR with its phase-shift logic is shown in Figure 2(c), along with the resulting output values. As can be verified, the bit sequence at each output stage has the appropriate phase shift with respect to the sequence of stage 0.

3. THE UNIFIED METHOD

The next state and output in a linear finite state machines (LFSM) (see, e.g., Gill [1966]) with n cells, l inputs, and m outputs are given by the relations

$$s_{t+1} = As_t + Bx_t, \quad y_t = Cs_t + Dx_t,$$

where s_t, x_t, y_t are column vectors representing respectively the state, input, and output at time t, and A, B, C, D are matrices of dimensions $n \times n$, $n \times l$, $m \times n$, and $m \times l$, respectively, known as characterizing matrices. Matrix A in particular is known as the *characteristic* or *transition matrix* of the LFSM. LFSMs used as TPG mechanisms have no external input (they are "autonomous") and therefore matrices B and D above are considered to be 0. (In the remaining we consider only autonomous LFSMs.) In traditional TPGs, that is, TPGs without phase shifters, the output is just the current state, that is, matrix C above is just the identity matrix I. The use of phase shifters results in effect in the replacement of the $n \times n$ identity matrix I with another $m \times n$ phase-shift matrix H. (Initially, we will assume that $m = n$.) For the examples in the previous section, the phase-shift matrix H is shown in Figure 1(b) and 2(c).

It is known that in order to obtain a sequence with phase shift p with respect to the sequence of stage k, $0 \leq k \leq d - 1$, for an LFSM with transition matrix $A = M$, the required XOR tap positions are indicated by the positions of the 1s in the kth row of M^p, that is, assuming that the XOR tap (row) vector is x and r_k is a (row) vector with a single 1 at position k, then

$$x = r_k M^p \tag{1}$$

(We make note of the fact that Equation (1), where r_k is a *row* vector, assumes that the next state equation of the LFSM is written—as in the definitions above—in the form $s_{t+1} = Ms_t$, where s_{t+1} and s_t are *column* vectors.)

Assuming (without loss of generality) that the reference stage is stage 0, and indicating by $R_i(A)$ the ith row of matrix A and by $|v|$ the number of 1s in vector v, relation (1) is presented in a more appropriate form in the lemma below:

Lemma 3.1. *Given an LFSM F of d stages with transition matrix M and bounds B and L, the corresponding phase-shift matrix H is obtained by putting as its ith row, the first (0th) row of each matrix* $M^{p_{i-1}}$, $1 \leq i \leq d$, *where* $p_i - p_{i-1} \geq L$, $p_0 = 0$, *and* $|R_0(M^{p_i})| \leq B$.

Theorem 3.1. *Given an LFSM F of d cells with transition matrix M and bounds B and L, the corresponding phase-shift matrix H is equal to*

$$H = \begin{bmatrix} \hat{s}_0 \\ \hat{s}_{p_1} \\ \cdots \\ \hat{s}_{p_{d-1}} \end{bmatrix},$$

where $(\hat{s}_{p_i})^{\mathrm{T}}$, $0 \leq i \leq d$, *are states of an LFSM with transition matrix* M^{T}, *starting with state* $(\hat{s}_0)^{\mathrm{T}} = [100 \cdots 0]^{\mathrm{T}}$, *and* $p_i - p_{i-1} \geq L$, $p_0 = 0$, $|\hat{s}_{p_i}| \leq B$.

PROOF. From Lemma 3.1 we have that the ith row of matrix H is equal to the 0th row of each matrix M^{p_i}. But for each satisfying p_i,

$$R_0(M^{p_i}) = R_0(M^{p_i-1} \cdot M) = R_{k+1}(M^{p_i-1}) \cdot M, \quad 1 \leq i \leq d-1,$$

where $R_0(M^0) = [100 \cdots 0]$. Considering each $R_0(M^{p_i})$ as a $1 \times d$ vector $\hat{s}_i$ we have that

$$\hat{s}_{p_i} = R_0(M^{p_i}) = R_0(M^{p_i-1}) \cdot M = \hat{s}_{p_i-1} \cdot M \Rightarrow$$

$$(\hat{s}_{p_i})^{\mathrm{T}} = M^{\mathrm{T}} \cdot (\hat{s}_{p_i-1})^{\mathrm{T}},$$

(A^{T} mean the transpose of matrix A), that is, the $1 \times d$ $R_1(M^{p_i})$ vectors can be seen as states ($d \times 1$ vectors) of an LFSM with transition matrix M^{T}, starting with state $(\hat{s}_0)^{\mathrm{T}} = [100 \cdots 0]^{\mathrm{T}}$. □

Theorem 3.1 provides the uniform basis to compute phase shifts for any LFSM with transition matrix M: simply simulate the LFSM with transition matrix M^{T} starting from state $[100 \cdots 0]^{\mathrm{T}}$. The actual runtime of this simulation depends on the phase-shift values requested and the size of the simulated LFSM. We observe that in the previous work of Ireland and Marshall [1968], matrix M^p in Equation (1) was proposed to be computed as the product of matrices Φ_p and Φ_0^{-1}, where Φ_0 is the matrix having as columns the states $s_0, s_1, \ldots, s_{d-1}$, where s_0 has a single 1 at position k, and Φ_p is the matrix having as columns the states $s_p, s_{p+1}, \ldots, s_{p+d-1}$. This is evidently much less efficient than the new procedure since matrix inversion for the computation of Φ_0^{-1} is required in the former. We also note that in the previous work of Mrugalski et al. [2000] for the special case of cellular automata, the computation of the product Φ_p and Φ_0^{-1} was avoided by substituting in Equation (1) M^{T} for M, since for cellular automata (but *not*, in general, for other LFSMs), $M = M^{\mathrm{T}}$.

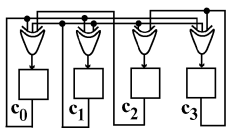
An illustration of the approach is given in Figure 3. An arbitrary LFSM (neither an LFSR or a CA) is shown in Figure 3(a) along with its transition matrix M and its state sequence. (The characteristic polynomial for this LFSM is $P(x) = x^4 + x^3 + 1$.) Assume that we want to obtain successive phase shifts that are at least $L = 4$ apart and can be attained with at most $B = 2$ taps per phase shifter. Then we consider the LFSM with transition matrix M^{T} (Figure 3(b)) and we simulate it starting with state [1000]. The vectors at cycles 0, 4, 10, 14 give the desired XOR tap positions (matrix H). The original LFSM plus the corresponding phase-shift logic is shown in Figure 3(c). As can be verified, the bit sequence of each output stage has the appropriate phase shift with respect to that of stage 0.

Theorem 3.1 generalizes the techniques previously proposed for the special cases of cellular automata and LFSRs.

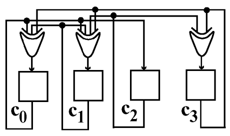
(i) For cellular automata, matrix M is symmetric and so M^{T} is the same as M. So the simulation of the LFSM with transition matrix M^{T} is actually the simulation of the original CA, as the existing technique for CA only [Mrugalski et al. 2000] specified.

(ii) For LFSRs, if the characteristic polynomial is $P(x) = c_d x^d + c_{d-1}x^{d-1} + \cdots + c_1 x + c_0$, where coefficient c_d is 1 as the LFSR has d cells and

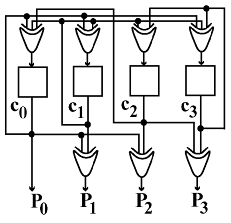
$$\mathbf{M} = \begin{bmatrix} 1&1&1&0\\ 1&1&0&0\\ 0&1&0&1\\ 1&1&0&1 \end{bmatrix}$$



$$\mathbf{M}^{\mathrm{T}} = \begin{bmatrix} 1&1&0&1\\ 1&1&1&1\\ 1&0&0&0\\ 0&0&1&1 \end{bmatrix}$$



$$\mathbf{M} = \begin{bmatrix} 1&1&1&0\\ 1&1&0&0\\ 0&1&0&1\\ 1&1&0&1 \end{bmatrix} \quad \mathbf{H} = \begin{bmatrix} 1&0&0&0\\ 1&1&0&0\\ 1&0&1&0\\ 0&0&1&1 \end{bmatrix}$$



Cyc	Stages 0 1 2 3
0	1 0 0 0
1	1 1 0 1
2	0 0 0 1
3	0 0 1 1
4	1 0 1 1
5	0 1 1 0
6	0 1 1 1
7	0 1 0 0
8	1 1 1 1
9	1 0 0 1
10	1 1 1 0
11	1 0 1 0
12	0 1 0 1
13	1 1 0 0
14	0 0 1 0

(a)

Cyc	Stages 0 1 2 3	
0	1 0 0 0	← 0
1	1 1 1 0	
2	0 1 1 1	
3	0 1 0 0	
4	1 1 0 0	← 4
5	0 0 1 0	
6	0 1 0 1	
7	0 0 0 1	
8	1 1 0 1	
9	1 1 1 1	
10	1 0 1 0	← 10
11	1 0 1 1	
12	0 1 1 0	
13	1 0 0 1	
14	0 0 1 1	← 14

(b)

Cyc	Stages 0 1 2 3
0	1 0 1 0
1	1 1 1 1
2	0 0 0 1
3	0 0 1 0
4	1 0 0 0
5	0 1 1 1
6	0 1 1 0
7	0 1 0 0
8	1 1 0 0
9	1 0 1 1
10	1 1 0 1
11	1 0 0 1
12	0 1 0 1
13	1 1 1 0
14	0 0 1 1

(c)

Fig. 3. Obtaining phase shifts for $L = 4$ and $B = 2$ for an arbitrary LFSM (characteristic polynomial $x^4 + x^3 + 1$). (a) Original LFSM with transition matrix M and state sequence. (b) LFSM with transition matrix M^{T} and state sequence starting with vector [1000]. (c) Original LFSM with the required phase shifters and resulting output sequence.

coefficient c_0 is 1 to make the LFSR nonsingular, then for $0 \le i \le d-1$, entry $M_{d-1,i} = c_i$ for a Type-1 LFSR (Figure 4(a)), whereas entry $M_{d-1-i,0} = c_i$ for a Type-2 LFSR (Figure 4(b)). All other entries of M in both cases are 0 except for entries $M_{i,i+1}$, $0 \le i \le d-2$, which are 1. Consider a Type-1 LFSR. The transpose L_1^{T} of its transition matrix L_1 is as shown in Figure 4(c). The transition matrix $L_{2,\tilde{P}}$ of a Type-2 LFSR with the reciprocal characteristic polynomial $\tilde{P}(x) = x^d P(\frac{1}{x}) = c_d + c_{d-1}x + \cdots + c_1 x^{d-1} + c_0 x^d = 1 + c_{d-1}x + \cdots + c_1 x^{d-1} + x^d$ is shown in Figure 4(d). As can be verified, $L_1^{\mathrm{T}} \cdot L_{2,\tilde{P}} = I$, that is, L_1^{T} is the inverse of $L_{2,\tilde{P}}$, and so the simulation of the LFSM with transition matrix L_1^{T} starting with state [1000], as the new algorithm specifies, is equivalent to the simulation of the dual LFSR $L_{2,\tilde{P}}$ starting with state [1000] and going in the reverse direction, as the existing technique for LFSRs only [Rajski et al. 1998] specified.

In applications such as avoiding linear dependencies in pseudorandom and pseudoexhaustive TPG [Bardell et al. 1987], or in test set embedding [Bellos et al. 2002], the stages of an LFSM mechanism are required to have exact phase shifts among them (see also Kagaris [2003]). We denote such phase shifts with

$$\mathbf{L}_1 = \begin{bmatrix} 0 & 1 & 0 & \dots & 0 & 0 \\ 0 & 0 & 1 & \dots & 0 & 0 \\ & & \dots & \dots & & \\ 0 & 0 & 0 & \dots & 1 & 0 \\ 0 & 0 & 0 & \dots & 0 & 1 \\ 1 & c_1 & c_2 & \dots & c_{d-2} & c_{d-1} \end{bmatrix} \qquad \mathbf{L}_2 = \begin{bmatrix} c_{d-1} & 1 & 0 & \dots & 0 & 0 \\ c_{d-2} & 0 & 1 & \dots & 0 & 0 \\ & & \dots & \dots & & \\ c_2 & 0 & 0 & \dots & 1 & 0 \\ c_1 & 0 & 0 & \dots & 0 & 1 \\ 1 & 0 & 0 & \dots & 0 & 0 \end{bmatrix}$$

(a) **(b)**

$$\mathbf{L}_1^{\mathbf{T}} = \begin{bmatrix} 0 & 0 & \dots & 0 & 0 & 1 \\ 1 & 0 & \dots & 0 & 0 & c_1 \\ 0 & 1 & \dots & 0 & 0 & c_2 \\ & & \dots & \dots & & \\ 0 & 0 & \dots & 1 & 0 & c_{d-2} \\ 0 & 0 & \dots & 0 & 1 & c_{d-1} \end{bmatrix} \qquad \mathbf{L}_{2,\tilde{P}} = \begin{bmatrix} c_1 & 1 & 0 & \dots & 0 & 0 \\ c_2 & 0 & 1 & \dots & 0 & 0 \\ & & \dots & \dots & & \\ c_{d-2} & 0 & 0 & \dots & 1 & 0 \\ c_{d-1} & 0 & 0 & \dots & 0 & 1 \\ 1 & 0 & 0 & \dots & 0 & 0 \end{bmatrix}$$

(c) **(d)**

Fig. 4. Relations among LFSR transition matrices (charcteristic polynomial $P(x) = x^d + c_{d-1}x^{d-1} + \cdots + c_1x + 1$). (a) Matrix of Type-1 LFSR. (b) Matrix of Type-2 LFSR. (c) Transpose of (a). (d) Matrix of Type-2 LFSR with reciprocal characteristic polynomial.

$[p_0, p_1, \dots, p_{d-1}]$. Finding exact phase shifts for an LFSM with transition matrix M would require simulating the LFSM with transition matrix M^{T} for $\max\{p_i\}$ cycles. However, this can be done very efficiently without the need of any LFSM simulation. Since $\hat{s}_{p_i} = \hat{s}_0 \cdot M^{p_i}$, each of the powers M^{p_i} can be computed fast by repeated squaring and summing of smaller powers. Moreover, since we are interested only in the first row of M^{p_i} (Lemma 3.1), the appropriate set of tap positions S can be found very fast by doing the following: given a requested phase shift with binary representation $p = b_{d-1} \dots b_1 b_0$, do repeatedly the following for all $b_i \neq 0$, $0 \leq i < d$: (i) set S to the XOR sum of all $R_{i,k}$, where $k \in F$ (F is initialized to $F = \{0\}$); and (ii) set F to the set of the positions of 1 in S.

Finally, we note that the stages of a TPG mechanism usually feed multiple scan chains in parallel (cf. STUMPS architecture [Bardell et al. 1987]). If the number m of scan chains that need to be driven is greater than the number d of the cells of the TPG, then extra phase shifters can be added to drive the remaining parallel scan chains. In this case the phase-shift matrix H is of dimension $m \times d$ and nothing else changes in its computation.

4. EXPERIMENTAL RESULTS-CASE STUDIES

We demonstrate the application of the proposed methods to compute phase shifters for two nontraditional classes of LFSMs used as TPG mechanisms. Such mechanisms can offer reduced propagation delay, short feedback wire length, and small gate fan-in and fan-out. The classes examined here are two-dimensional cellular automata (see, e.g., Cattell et al. [1999]) and ring generators [Rajski et al. 2002].

4.1 Two-Dimensional Cellular Automata

An example of a 2-by-16 cellular automaton is given in Figure 5(a). Each box in Figure 5(a) represents a flip-flop storing the result of an XOR gate whose

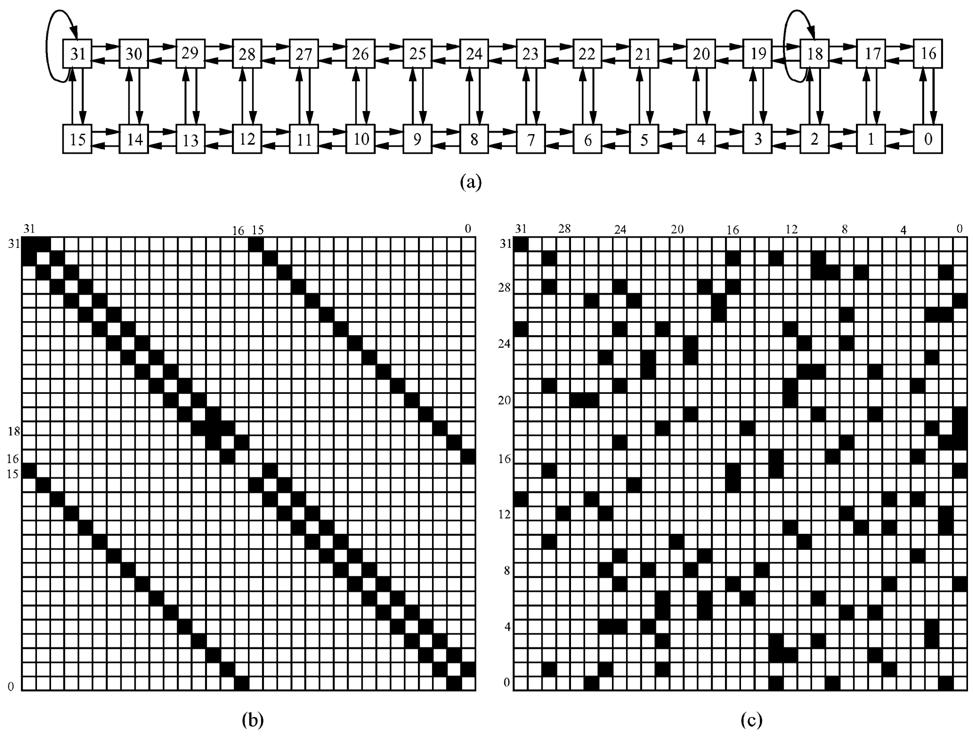
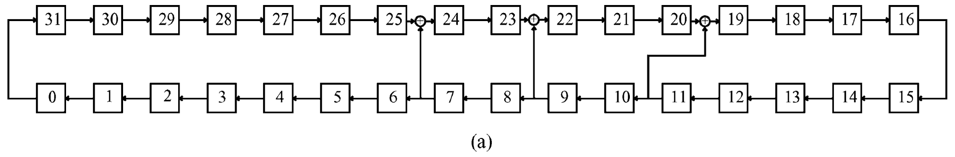


Fig. 5. (a) 2×16 CA. (b) Transition matrix. (c) Phase-shift matrix.

inputs are shown by the arrows. The corresponding transition matrix is given in Figure 5(b). (Black (white) squares indicate entries with 1 (0), respectively.) The general form of the transition matrix of a 2-by-n CA is a $2n \times 2n$ matrix whose upper right and lower left $n \times n$ submatrices are the identity matrix, the first lower and first upper subdiagonals of the upper left and lower right $n \times n$ submatrices are all 1s, and the main diagonal of the whole matrix coincides with the rule vector determining the operation of the 2-by-n CA. For the example we used the rule vector that has a 1 at positions 31 and 18 (taken from the list in Cattell et al. [1999] after aligning index i with $2n - i$). This results in a maximum length sequence with (primitive) characteristic polynomial $p(x) = x^{32} + x^{30} + x^{29} + x^{26} + x^{24} + x^{21} + x^{20} + x^{18} + x^{17} + x^{13} + x^{12} + x^{6} + x^{5} + x^{4} + x^{3} + x + 1$.

As can be observed, the transition matrix of Two-dimensional CA is symmetric, so the simulation is actually done on the original machine, as was the case for one-dimensional CA. Assume that we want to obtain successive phase shifts that are at least $L = 5000$ and can be attained with at most $B = 4$ taps per phase shifter. Then by applying the method in Section 3 we find the phase-shift matrix shown in Figure 5(c). The corresponding actual phase shifts attained, given here as differences between successive stages rather than cumulative values with respect to stage 0, are 161300, 16202, 165974, 81719, 40926, 36944, 13003, 52745, 77226, 22707, 26522, 215151, 254904, 30018, 90915, 131904, 150865,



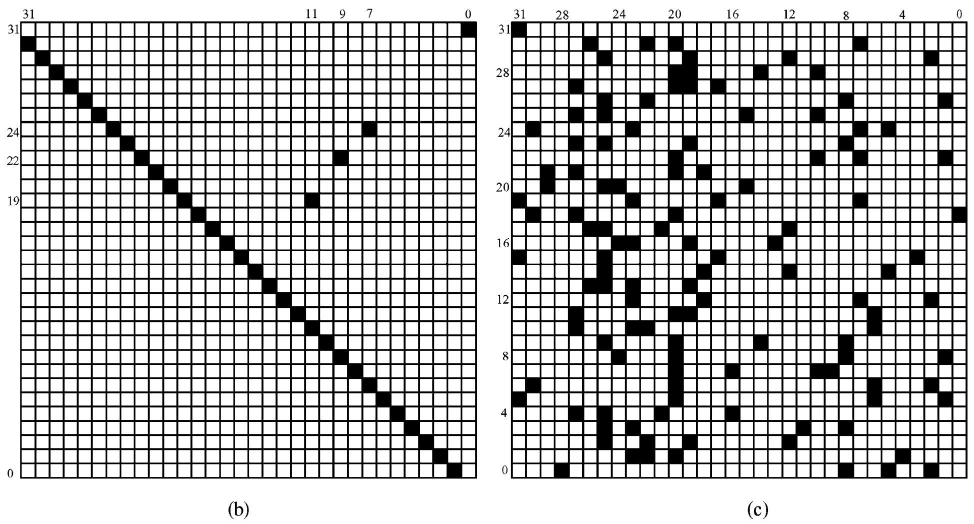


Fig. 6. (a) 32-bit ring generator. (b) Transition matrix. (c) Phase-shift matrix.

144616, 12612, 89195, 111158, 280637, 199848, 73328, 286842, 296954, 31052, 5413, 167602, 15121, 11934. The time to obtain these phase shifts on a SUN Blade 1500 was 34.88 s.

4.2 Ring Generators

An example of a 32-bit ring generator along with its transition matrix are shown in Figures 6(a) and 6(b), respectively. The characteristic polynomial is $p(x) = x^{32} + x^{23} + x^{18} + x^{14} + 1$. This example is the same as in Rajski et al. [2002], but with the reciprocal characteristic polynomial. Assume that we want to obtain successive phase shifts that are again at least $L = 5000$ and can be attained with at most $B = 4$ taps per phase shifter. Then by applying the method in Section 3 we find the phase shift matrix shown in Figure 6(c). The corresponding actual phase shifts attained, given here again as differences between successive stages, are: 290686, 57635, 9455, 408742, 684521, 892995, 159529, 815372, 312894, 344026, 239408, 501746, 112685, 177406, 357799, 146916, 967607, 160863, 870413, 864915, 405443, 55536, 81402, 163621, 932622, 1009405, 171816, 348967, 163779, 124078, 68344. The time to obtain these phase shifts on a SUN Blade 1500 was 85.7 s.

5. CONCLUSIONS

We presented a unified method to find phase shifters for any LFSM. The method carries out efficiently the fundamental step of simulation-based search strategies to find phase-shift vectors satisying a given tap bound. As a demonstration, we showed how phase shifters can be computed for two-dimensional CA and ring generators.

REFERENCES

BARDELL, P. H., MCANNEY, W. H., AND SAVIR, J. 1987. *Built-in Test for VLSI*. Wiley, New York, NY.

BELLOS, M., KAGARIS, D., AND NIKOLOS, D. 2002. Test set embedding based on phase shifters. In *4th European Dependable Computing Conference, October 2002*. Lecture Notes in Computer Science, vol. 2485, Springer-Verlag, Berlin, Germany, 90–101.

CATTELL, K., ZHANG, S., SERRA, M., AND MUZIO, J. C. 1999. 2-by-*n* Hybrid cellular automata with regular configuration: Theory and applications. *IEEE Trans. Comput. 48*, 3, 285–295.

GILL, A. 1966. *Linear Sequential Circuits*. McGraw-Hill, New York, NY.

IRELAND, B. AND MARSHALL, J. E. 1968. Matrix method to determine shift-register connections for delayed pseudorandom binary sequences. *Electron. Lett. 4*, 15, 309–310.

KAGARIS, D. 2003. Built-in TPG with designed phaseshifts. In *Proceedings of the IEEE VLSI Test Symposium* (April). 365–370.

MRUGALSKI, G., RAJSKI, J., AND TYSZER, J. 1999. Comparative study of CA-based PRPGs and LFSRs with phase shifters. In *Proceedings of the VLSI Test Symposium*. 236–245.

MRUGALSKI, G., RAJSKI, J., AND TYSZER, J. 2000. Cellular automata-based test pattern generators with phase shifters. *IEEE Trans. CAD/ICAS 19*, 8, 878–893.

RAJSKI, J., TYSZER, J., KASSAB, M., ET AL. 2002. Embedded deterministic test for low cost manufacturing test. In *Proceedings of the International Test Conference*. 301–310.

RAJSKI, J., TAMARAPALLI, N., AND TYSZER, J. 1998. Automated synthesis of large phase shifters for built-in self-test. In *Proceedings of the International Test Conference*. 1047–1056.

Received July 2004; revised August 2004; accepted August 2004

An Efficient Algorithm for Finding the Minimal-Area FPGA Technology Mapping

CHI-CHOU KAO
National Pingtung Institute of Commerce
and
YEN-TAI LAI
National Cheng Kung University

Minimum area is one of the important objectives in technology mapping for lookup table-based field-progrmmable gate arrays (FPGAs). Although there is an algorithm that can find an optimal solution in polynomial time for the minimal-area FPGA technology mapping problem without gate duplication, its time complexity can grow exponentially with the number of inputs of the lookup-tables. This article proposes an algorithm with approximate to the area-optimal solution and lower time complexity. The time complexity of this algorithm is proven theoretically to be bounded by $O(n^3)$, where n is the total number of gates in the given circuit. It is shown that except for some cases the proposed algorithm can find an optimal solution of a given problem. We have combined the proposed algorithm with the existing postprocessing procedures which are used to find the gates that can be duplicated on a set of benchmark examples. The experimental results demonstrate the effectiveness of our algorithm.

Categories and Subject Descriptors: B.6.1 [**Logic Design**]: Design Styles—*Combinational logic*; B.6.3 [**Logic Design**]: Design Aids—*Automatic synthesis*; *optimization*; J.6 [**Computer-Aided Engineering**]: *Computer-aided design (CAD)*

General Terms: Algorithms, Design, Theory, Performance

Additional Key Words and Phrases: Logic synthesis, technology mapping, partition, greedy method

1. INTRODUCTION

The merits of low cost and short turnaround time have made field-programmable gate arrays (FPGAs) an important technology in very-large-scale integration (VLSI) designs. In an FPGA device, a configurable logic block

This research was supported by the National Science Council (NSC) of R.O.C. under Contracts 93-2215-E-251-001 and 93-2215-E-006-018.

Authors' addresses: C.-C. Kao, Department of Information Technology, National Pingtung Institute of Commerce, Pingtung, Taiwan, 900; email: cckao@casdc.ee.ncku.edu.tw; Y.-T. Lai, Department of Electrical Engineering, National Cheng Kung University, Tainan, Taiwan, 701; email: ytlai@casdc.ee.ncku.edu.tw.

(CLB) contains a k-input lookup table (LUT) and can implement any Boolean function whose input number does not exceed k, $k$being a function of the CLB hardware constraints on the number of inputs.

The technology mapping problem for LUT-based FPGAs is to produce an equivalent circuit for a given circuit using gates that can be implemented with LUTs. Given a circuit, if the number of distinct inputs of a gate is more than k, then no LUTs can include this gate. Therefore, such a gate must be decomposed into two or more gates, which are functionally equivalent to the original gate, so as to reduce the number of inputs of a gate before the technology mapping step is performed. Some previously developed decomposition techniques find important applications in LUT-based FPGAs [Huang et al. 1994, 1995; Karp and Roth 1962; Lai et al. 1993; Murgai et al. 1991; Sawada et al. 1995; Shen et al. 1995; Wurth et al. 1995].

The optimization objectives of the technology mapping tasks include area minimization, delay optimization, routability maximization, or a combination of these. For the area minimization objective, if duplication of gates is allowed, the existing algorithms can be classified into three categories: (1) those using enumeration methods including Chortle [Francis et al. 1990], Chortle-crf [Francis et al. 1991], MIS-pga [Murgai et al. 1990], MIS-pga (new) [Murgai et al. 1991], Vismap [Woo 1991], and MILP [Chowdhary and Hayes 1995] algorithms; (2) those using heuristic algorithms, for example, the Level-Map algorithm [Farrahi and Sarrafzadeh 1994], and the CutMap algorithm [Cong and Hwang 1995] etc.; and (3) those combining decomposition with mapping algorithms including the Xmap algorithm [Karplus 1991], the FGSyn algorithm [Lai et al. 1994], the TechMap algorithm [Sawkar and Thomas 1992], the method developed by Lehman et al. [1997], and the SLDMap algorithm [Chen and Kong 2001]. Proper duplication of gates can be beneficial to area minimization. However, excessive duplication of gates adds to the number of gates such that the total number of LUTs can be increased. The problem, which is to find the minimal-area FPGA technology mapping with gate duplication, is NP-complete [Farrahi and Sarrafzadeh 1994; Zhang et al. 1996]. Using a two-stage design process, it is possible to first find a solution without gate duplication and then find the part that can be duplicated to reduce the total number of LUTs.

By using the dynamic programming approach to generate cuts, the DFmap [Cong and Ding 1994b] found an optimal solution in polynomial time for the minimal-area FPGA technology mapping problem without gate duplication. However, the cut generation step is bounded by $O(n^k)$, where n is the total number of gates and k is the inputs number of LUTs in the given circuit [Cong et al. 1999]. The DFmap can take a large amount of run time for some homogeneous FPGAs consisting of LUTs with large input size.

In this article, we develop an efficient area-minimum mapping algorithm without gate duplication for LUT-based FPGAs. This proposed algorithm first uses a partitioning algorithm that divides the given circuit system into subsystems such that one subsystem has only one output and the union of the solutions for these subsystems is the solution of the whole system. Then a greedy approach is proposed to find the mapping solution for each subsystem. Two kinds of subgraphs of the graph representing a subsystem are selected, one at a time, in the procedure of the greedy method. It is shown that except for a few circuits this method can find the optimal mapping of the subsystem. The time complexity of this algorithm is proven theoretically to be bounded by $O(n^3)$, where n is the total number of gates in the given circuit. According to the experimental results on a number of MCNC benchmarks as shown in Section 6, it can be seen that the proposed algorithm takes much less CPU time than the DFmap, and that the number of LUTs is very close to the optimal-area mapping solution found by the DFmap. The proposed algorithm can also be used to find the area-minimum mapping solution for a heterogeneous FPGA because the number of inputs of the LUTs does not affect the time complexity of the algorithm. Moreover, we carry out the existing postprocessing procedures [Cong and Ding 1994a] which are used to find the gates that can be duplicated to reduce further the total number of LUTs. Compared with the existing algorithm [Cong and Ding 1994a; Francis et al. 1991; Cong and Hwang 1995] with gate duplication, the experimental results demonstrate the efficiency of this proposed algorithm.

The remainder of this paper is organized as follows. Section 2 describes the terminology and problem formulation. The outline of the area-minimal algorithm for duplication-free mapping is presented in Section 3, and the algorithm is described in Sections 4 and 5. Experimental results are shown in Section 6. Finally, in Section 7 we present concluding remarks.

2. TERMINOLOGY AND PROBLEM FORMULATION

A combinational logic circuit can be represented by a directed acyclic graph (DAG), $G = (V, E)$. A vertex in V represents a logic gate or a primary input/output, and a directed edge $\langle i, j \rangle$ exists in E if the output of gate i is the input of gate j. A primary input (PI) vertex has no incoming edge and a primary output (PO) vertex has no outgoing edge. Let v and u be two vertices of V. If there is a directed path from v to u, v is said to be a *predecessor* of u and u is a *successor* of v. It is defined that *input*(v) represents the set of vertices which are the fan-ins of v. Similarly, *output*(v) represents the set of vertices which are the fan-outs of v. The output signals from a vertex which has more than one out-edge propagate through distinct paths and end at (1) more than one PO vertex or (2) one vertex. The vertices are called *multiple-fan-out sources* in the first case and *reconvergent sources* in the second case. For example, Figure 1(b) shows the corresponding DAG of the combinational circuit illustrated in Figure 1(a). In Figure 1(b), u and x are multiple-fan-out sources; s is a reconvergent source. Notice that a reconvergent source is not a multiple-fan-out source.

Assume C_v is the subgraph induced by the set of non-PI vertices, V_v. The subgraph C_v is said to be a *cone* if there exists a vertex $v \in V_v$ such that, for every vertex $u \in C_v$ there is a directed path from u to v in C_v. The vertex v is called the *root* of the cone. Since C_v is the induced subgraph of V_v, the set of fan-in signals of C_v is exactly equal to $|input(V_v)|$. For convenience, $input(V_v)$ and $input(C_v)$ are used simultaneously and interchangeably in the rest of this article. A *fan-out-free cone* (*FFC*) is a cone in which the fan-outs of every vertex other than the root are in the cone. A cone, C_v, is said to be *k-feasible* if it is a

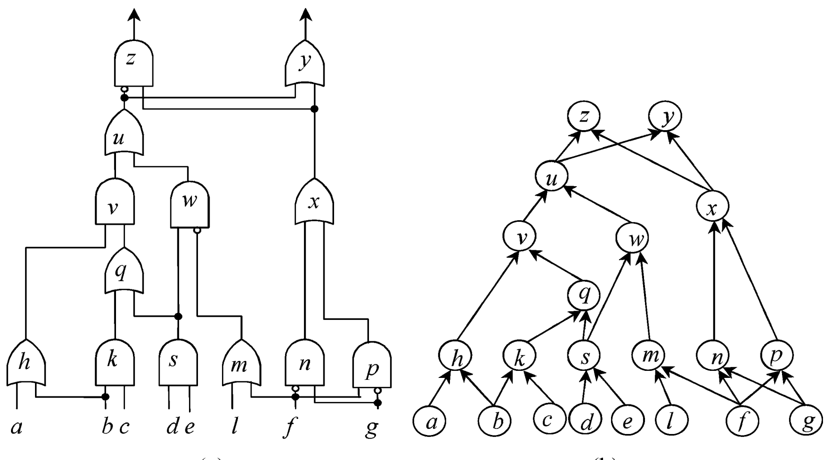


Fig. 1. (a) A combinational circuit; (b) the DAG corresponding to the circuit in (a).

FFC and $|input(C_v)|.k$. A FFC rooted at v, C_v, is called a *primary block* if it covers all FFCs rooted at v and v is either a PO vertex or a multiple-fan-out source.

Assume C_v is the subgraph induced by the set of non-PI vertices, V_v. The subgraphC_v is said to be a *cone* if there exists a vertex $v \in V_v$ such that for every vertex $u \in C_v$ there is a directed path from u to v in C_v. The vertex v is called the *root* of the cone. Since C_v is the induced subgraph of V_v, the set of fan-in signals of C_v is exactly equal to $|input(V_v)|$. For convenience, $input(V_v)$ and $input(C_v)$ are used simultaneously and interchangeably in the rest of this article. A *fan-out-free cone* (*FFC*) is a cone in which the fan-outs of every vertex other than the root are in the cone. A cone, C_v, is said to be *k-feasible* if it is a FFC and $|input(C_v)|.k$. A FFC rooted at v, C_v, is called a *primary block* if it covers all FFCs rooted at v and v is either a PO vertex or a multiple-fan-out source.

A network is said to be *k-bounded* if the in-degree of every vertex is less than or equal to k in the network. In the rest of this article, the given DAG is transformed into a two-input simple gate network to ensure that every gate can be realized by a LUT [Cong et al. 1992]. The reason is that gates with a smaller number of inputs can easily be included in a LUT and may increase the possibility of obtaining a better mapping solution. Let $C_u = (V_u, E_u)$ and $C_v = (V_v, E_v)$ be two cones. C_u is said to be *covered* with C_v if $V_u \subseteq V_v$. Formally, a collection of k-feasible cones is said to be a *mapping solution* of $G = (V, E)$ if each vertex in V is covered in one and only one cone. Therefore, the technology mapping problem for LUT-based FPGA designs can be formulated as a graph-covering problem as follows:

Given a k-bounded network, find a mapping solution with the minimum number of cones.

3. OUTLINE OF THE MAPPING ALGORITHM

In this article, we present an efficient algorithm in which gates are implemented by one and only one LUT for finding the area-minimum solution. The proposed algorithm is divided into two steps: (1) partitioning the graph representing the given circuit system into primary blocks, and (2) using the greedy method to find the solution for each primary block.

The divide-and-conquer approach is usually used to reduce the complexity of computation. To use this approach, we must be able to partition the given problem into subproblems such that each subproblem can be solved independently and the solutions for the subproblems can be combined to be the solution of the whole problem. An algorithm for partitioning the given graph is shown in Section 6. It traverses in the breadth-first order and labels the vertices in the graph such that the vertices in each primary block have the same label.

To use the greedy method to find the optimal mapping for a primary block, G_B, we must find a k-feasible cone C_t such that the optimal solution of G_B is the union of $\{C_t\}$ and the optimal solution of $G_B - C_t$. Subgraphs are selected in the iterations of the greedy procedure. It is shown that in an iteration the selection of a subgraph of a special kind can always lead to an optimal solution. However, a subgraph of this special kind may not exist in the selection procedure. In such a case, a subgraph of the second kind is selected. It is shown that except for some particular cases the selection of the second kind can also lead to finding the optimal solution. Algorithms for finding the subgraphs by the greedy method are described in Section 5.

4. PARTITIONING THE GIVEN DAG INTO PRIMARY BLOCKS

It is shown in this section that the union of the mappings of the primary blocks is the mapping of the given DAG. We must prove three aspects: (1) primary blocks are mutually exclusive, (2) the union of all primary blocks includes every vertex in the given DAG, and (3) every feasible cone is covered with exactly one primary block. An algorithm for partitioning the given graph into primary blocks is then presented.

LEMMA 1. *Primary blocks of a given DAG must be mutually exclusive.*

PROOF. Assume $C_s = (V_s, E_s)$ and $C_t = (V_t, E_t)$ are two primary blocks and C_s and C_t are not mutually exclusive. Let $V_r = V_s \cap V_t$ and u be a vertex in V_r. We denote s and t as the roots of C_s and C_t, respectively. A path from u to a PO vertex must pass through both s and t. Assume all these paths pass through s first and then t. Since every path starting from s must pass through t, every output signal from s ends at t. Equivalently, s is a reconvergent source but it is not a multiple-fan-out source. Therefore, C_s is not a primary block. This conflicts with the assumption that C_s is a primary block. □

LEMMA 2. *The union of all primary blocks includes all non-PI vertices in V.*

PROOF. Let u be a non-PI vertex in V. We consider two cases with u: (1) the paths starting from u end at more than one PO vertex and (2) all the paths starting from the vertex u end at one and only one PO vertex v. In the first case, there are two subcases: (1) u is a multiple-fan-out source, and (2) there is a multiple-fan-out source w such that all paths starting from u pass through w. In the first subcase, the subgraph induced by $\{u\}$ is an FFC and it is included in the primary block rooted at u. In the second subcase, the subgraph induced by the vertices on the paths from u to w is an FFC included in the primary block rooted at w.

Next, consider the second case, the subgraph defined by the vertices on these paths is an FFC rooted at v. According to the definition of a primary block, this cone is included in the primary block rooted at v. □

THEOREM 3. *For an FFC, C_k, there is one and only one primary block, C_w, such that $C_k \subseteq C_w$.*

PROOF. To prove this theorem, based on Lemmas 1 and 2, it remains to be proven that an FFC, $C_k = (V_k, E_k)$, is included in a single primary block, $C_w = (V_w, E_w)$. This proof is presented as follows:

If C_k is a primary block, $C_k = C_w$, then the theorem is proved. Otherwise, according to Lemma 2, C_k must be included in some primary blocks. Assume that the number of the primary blocks including C_k is more than one and C_w is one of those primary blocks. In other words, the vertices in C_k are not all included in C_w. Let $V_h = V_k \cap V_w$ and k and w be the roots of C_k and C_w, respectively. There are four cases with k and w: (1) $k \in V_h$ and $w \in V_h$, (2) $k \notin V_h$ and $w \in V_h$, (3) $k \in V_h$ and $w \notin V_h$, and (4) $k \notin V_h$ and $w \notin V_h$, as shown in Figure 2.

In the first case, since C_w is an FFC, the subgraph induced by V_h must be an FFC rooted at w. Hence, $k = w$. However, since C_w includes all FFCs rooted at $w, V_k \subset V_w$. It is a contradiction to the hypothesis that the vertices in V_k are not all included in V_w. Consider the second case. Recall that w is either a PO vertex or a multiple-fan-out source. Since $w \in V_k$ and $w \neq k$, w must be a multiple-fan-out source. There is a path passing through a vertex other than k. Hence, C_k is not an FFC, which is in contradiction to the assumption. In the third case, the cone including all the vertices in $V_k \cup V_w$ is an FFC rooted at w. This is a contradiction to the hypothesis that C_w is a primary block, which includes all FFCs rooted at w. In the last case, neither C_w nor C_k is an FFC. The theorem is therefore proven. □

According to Theorem 3, we can partition the given DAG into primary blocks and find a mapping solution for each primary block. The union of the mapping solutions of all primary blocks is the mapping solution of the given DAG.

To partition the given DAG into primary blocks, we label the vertices in the given DAG such that the vertices in a primary block have the same label. Starting from the PO vertices, a traversal of the graph in topological order can label the vertices. A vertex is labeled with the same tag as its fan-out vertex, if all its fan-out vertices have the same label; it is labeled with new tag otherwise. Since a reconvergent source has more one fan-out vertex, the *timesVisited*(u) is used to count the number of times that u has been visited such that every reconvergent source is labeled correctly. Figure 3 shows the result of labeling vertices of the graph in Figure 1(b). It is seen that in Figure 3 the fan-out vertices of vertex u have different labels and therefore vertex u is labeled with a new tag.

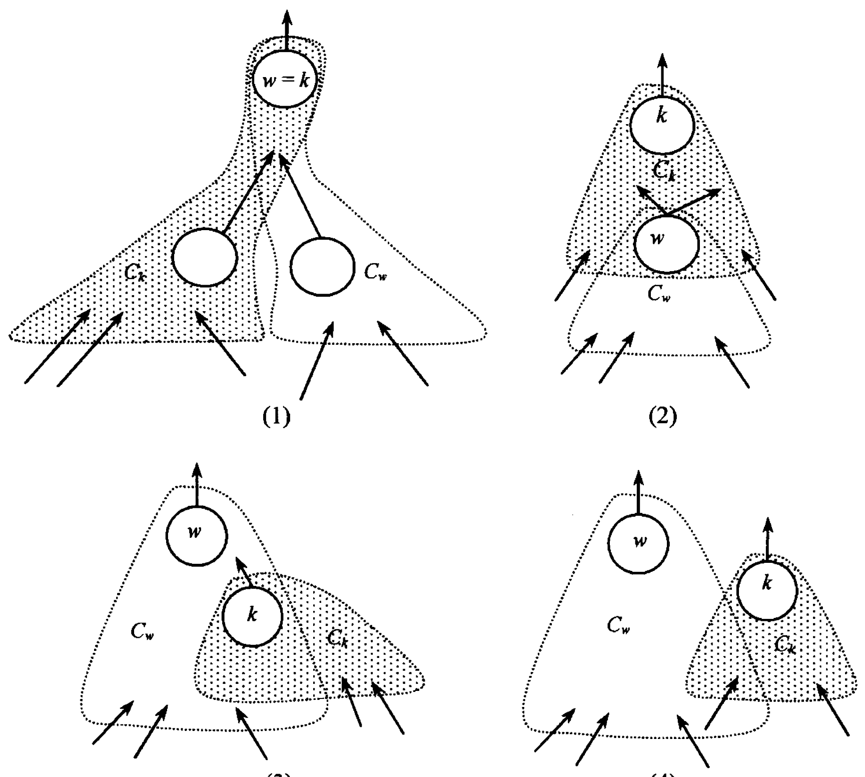


Fig. 2. All cases in which an FFC is not totally included in a primary block.

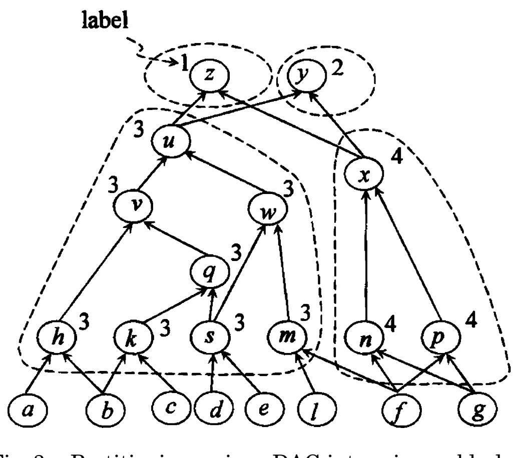


Fig. 3. Partitioning a given DAG into primary blocks.

The procedure to partition the given DAG into primary blocks is as follows:

```
Algorithm Primary Block Partitioning:
  PrimaryBlock_Partitioning ($G$)
  Comment: $G$ is a DAG
  begin
    for every vertex $v$ do
      if $v$ is a PO vertex begin
        label $v$ with a new tag;
        enqueue($v$, $Q$);
      end
    while $Q \neq \oslash$ begin
    $v \leftarrow$ dequeue ($Q$);
      for every fan-in vertex of $v$, $u$, do begin
        if $u$ has not been labeled begin
          label $u$ with the same tag as $v$;
          increment timesVisited($u$);
        end
        else if $u$ and $v$ have the same tag // can be a reconvergent source
          increment timesVisited($u$);
        else if $u$ is not in the queue $Q$ begin
          //fan-out vertices have different tags,
          //$u$ is a new found multiple vertex
          label $u$ with a new tag;
            enqueue($u$, $Q$);
          end
          if (timesVisited($u$) is equal to the outdegree of $u$) and
            ($u$ is not in the queue $Q$)
            enqueue($u$, $Q$);
        end of for loop
      end of while loop
    end of PrimaryBlock_Partitioning
```

According the labeling rule, a vertex that is labeled with a new tag must be a PO vertex or a multiple-fan-out source. Hence, the subgraph induced by the vertices with the same label must be an FFC rooted at a PO vertex or a multiple-fan-out source. Therefore, the subgraph induced by the vertices with the same label is a primary block. In this procedure, every edge is traversed once. The time complexity is bounded by $O(e)$.

5. THE MAPPING SOLUTION FOR A PRIMARY BLOCK

In this subsection, we will use the greedy method to find the mapping solution for a primary block, $G_B = (V_B, E_B)$. It is discussed whether the selection of a k-feasible cone in each iteration of the greedy method will lead to finding the optimal mapping solution or not.

A cone is called a *floor cone* if all its fan-in vertices are PI vertices. In Figure 4(a), the cone including the vertex r and all its non-PI predecessors is a floor cone.

THEOREM 4. *There exists an optimal mapping solution, M_B, in G_B such that every feasible floor cone is covered with a cone in M_B.*

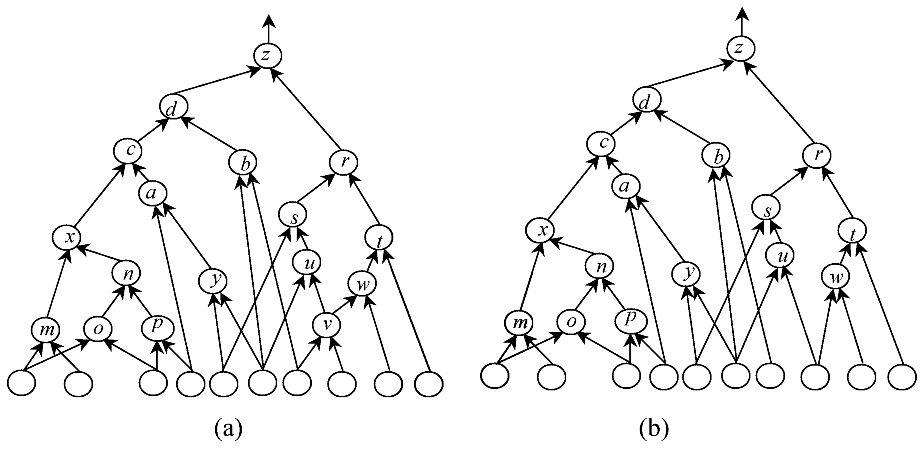


Fig. 4. (a) A given DAG ; (b) a new DAG derived from (a) by selecting a critical floor cone as a cone in the optimal mapping solution ($k = 5$).

Proof. Assume that C_w is a feasible floor cone tipped at w and that it is not covered with a single cone in an optimal mapping solution M'_B. Let C_v be the cone in M'_B that covers w. There are two cases with v and w: (1) $v = w$ and (2) $v \neq w$. Consider the first case. Assume $C_v \neq C_w$. Because C_w includes all non-PI predecessors of w, $C_v \subset C_w$. Let $G_v = C_w - C_v$ and S_v be the set of cones covering G_v in M'_B. It is obvious that $\cup_{C \in S_v} C = G_v$. If we replace C_v and all the cones in S_v with C_w, the number of cones in the new mapping solution decreases. This is a contradiction to the assumption that M'_B is an optimal mapping solution. Therefore, if $v = w$, then $C_v = C_w$. The second case is shown in Figure 5. Let $C'_w = C_w \cap C_v$, $G_u = C_w - C'_w$, and $C'_v = C_v - C'_w$. Since w is the only vertex in C_w that has fan-out signal to C'_v and $|Output(G_u)| \geq 1$, $|Input(C'_v)| = |Input(C_v)| - |Output(G_u)| + 1 \leq |Input(C_v)|$. Therefore, C'_v is a k-feasible cone. Let S_C be the set of cones covering G_u in M'_B. Since $|Output(G_u)| \geq 1$, $|S_C| \geq 1$. Accordingly, if we replace C_v and the cone in S_C with C'_v and C_w, the total number of cones in M'_B does not increase. According to the above discussions, we can replace M'_B with another optimal mapping solution, M_B, in G_B such that every feasible floor cone is covered with a cone in M_B. □

Let C_v be a k-feasible floor cone tipped at v. If the floor cone tipped at every successor of v is not k-feasible, C_v is called a *saturated floor cone*. In Figure 4(a), the floor cone including the set of the vertices $\{x, m, n, o, p\}$ is an example of a saturated floor cone. The floor cone including the set of the vertices $\{n, o, p\}$ is a floor cone but not a saturated floor cone because x is a successor of n and the floor tipped at x is a k-feasible floor cone. A saturated floor cone C_v is said to be a *critical floor cone* if it cannot be covered with any other k-feasible cones. For example, in Figure 4(a), C_v including the vertex v is a critical floor cone. On the other hand, the floor cone including the vertices $\{x, m, n, o, p\}$ is a saturated floor cone but not a critical floor cone in Figure 4(a).

Theorem 5. *If there exists a critical floor cone in G_B, it can be selected to be in the optimal mapping solution.*

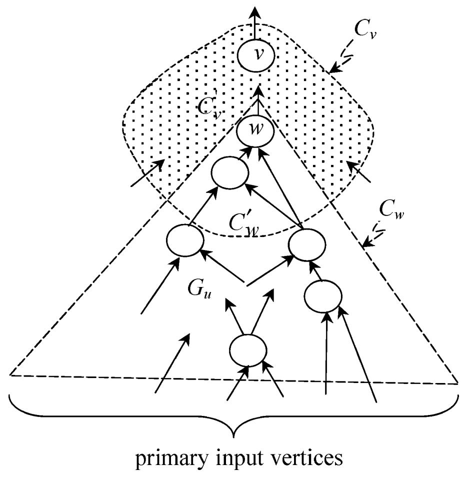


Fig. 5. A feasible floor cone is not covered with a cone in an optimal mapping solution in G_B.

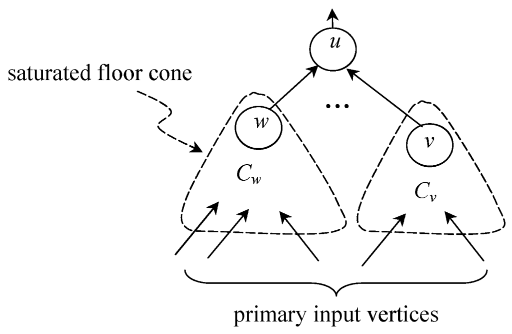


Fig. 6. Every floor cone tipped at a vertex in U_i is feasible.

PROOF. This is a corollary of Theorem 4. Assume C_r is a critical floor cone. According to Theorem 4, there exists an optimal mapping solution that has a cone, C_s, covering C_r. Since no feasible floor cone is larger than the critical floor cone, $C_s = C_r$. □

According to Theorem 5, the critical floor cone C_v in Figure 4(a) can be selected to be in the optimal mapping solution. On the other hand, if there are no critical floor cones, we must find another kind of cone that can be selected to be in the mapping solution. A vertex is called a *leading vertex* if (1) each fan-in vertex is either the tip of a saturated floor cone or a PI vertex, and (2) at least one fan-in vertex is the tip of a saturated floor cone. For example, in Figure 4(b) the vertex c is a leading vertex.

LEMMA 6. *There must exist a leading vertex in G_B.*

PROOF. We use induction on $|V_B|$ to prove the claim for all $|V_B| \in \mathbf{N}$. Obviously, there must exist a leading vertex for $|V_B| = 2$.

Suppose that there exists a leading vertex with $|V_B| = 2, 3, \ldots, k$. Given a DAG, $G'_B = (V'_B, E'_B)$, where $|V'_B| = k + 1$. As shown in Figure 6, let u be a fan-out vertex of a saturated floor cone C_w and U_i be the set of fan-in vertices of u. It is to be shown that if every floor cone tipped at a vertex in U_i is feasible, u is a leading vertex. Otherwise, there must exist a leading vertex that is a predecessor of u.

We consider two cases with U_i: (1) every floor cone tipped at a vertex in U_i is feasible and (2) at least one floor cone tipped at a vertex in U_i is infeasible. Consider the first case. Let $v \in U_i$, $v \neq w$, and C_v be a feasible floor cone tipped at v. If v is not a multiple-fan-out source, the successor of w is also the successor of v. Since C_w is a saturated floor cone, the floor cone tipped at a successor of v is not feasible. Hence, C_v is a saturated floor cone. On the other hand, v can be a multiple-fan-out source. Assume C_v is not a saturated floor cone and that there exists a saturated floor cone, C_x, tipped at a successor of v. Obviously, C_x must cover u and C_w. However, it is a contradiction to the hypothesis that C_w is a saturated floor cone and that it cannot be covered with any other saturated floor cones. Therefore, C_v is a saturated floor cone and u is a leading vertex.

Next we consider the second case. Let $v \in U_i$ and C_v be an infeasible floor cone covering v and all its non-PI predecessors. If u is not a leading vertex, there must in G_B exist an infeasible floor cone C_v. Clearly, the total number of vertices in C_v must be less than $k + 1$. According to the hypothesis that there exists a leading vertex with $|V_B| = 3, 4, \ldots, k$, there is a leading vertex in C_v. By the principle of induction, the claim holds for every $|V_B| \in \mathbf{N}$. □

Let S_v be the set of saturated floor cones whose fan-out vertex is a leading vertex v. A cone $C_i \in S_v$ is called the *prime cone* if $|Input(C_i)|$ is the largest in S_v. A leading vertex is said to be *critical* if there exists no feasible cone covering the prime cone and any other saturated floor cones in S_v. For example, in Figure 4(b), the vertex c is a critical leading vertex.

THEOREM 7. *If there are no critical floor cones, then a prime cone whose fan-out vertex is a critical leading vertex can be selected to be in the optimal mapping solution.*

PROOF. By Lemma 6, there must exist a leading vertex. Let C_w be a prime cone tipped at w and its fan-out vertex be a critical leading vertex, x. Since C_w is a feasible floor cone, according to Theorem 4, there exists an optimal mapping solution M_C which has a cone C_v covering C_w. Let v be the tip of C_v. Since no critical floor cones exist, $v \neq w$. Hence, there is a path from w to v. Equivalently, v is a successor of w. As in Figure 7, assume that x has a fan-in signal from a feasible floor cone C_u and $|Input(C_u)| \leq |Input(C_w)|$. Since x is a critical leading vertex, there exist no feasible cones covering both C_u and C_w. In other words, C_u must be a cone in M_C. Let $C'_v = C_u \cup (C_v - C_w)$. $|Input(C'_v)| = |Input(C_u)| + 1 \leq |Input(C_w)| + 1 = |Input(C_v)| \leq k$. Hence, we can obtain a new mapping solution by replacing C_v and C_u with C_w and C'_v. The new mapping solution will have the same number of cones. Equivalently, the new mapping solution is also an optimal one. □

According to Theorems 5 and 7, the greedy method can be used to find the optimal solution. Figure 8 illustrates the procedure of finding an area-optimal mapping solution. We first find the leading vertex c and the prime cone C_x

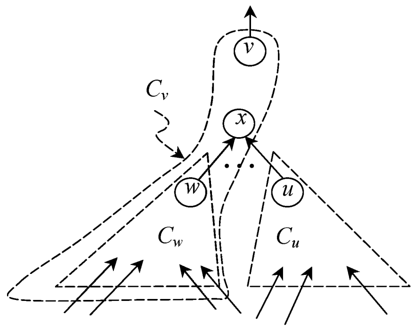


Fig. 7. The prime cone whose fan-out vertex is a critical leading vertex in the optimal mapping solution.

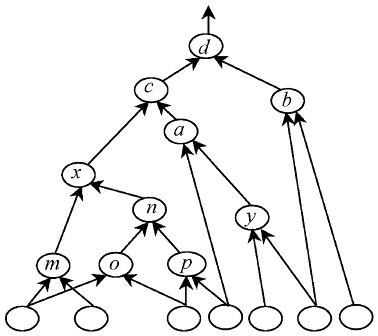


Fig. 8. An example of finding an optimal mapping solution by the greedy method ($k = 4$).

including the set of the vertices $\{x, m, n, o, p\}$. The cone C_x is selected to be in the mapping solution. A new DAG $G_B = G_B - C_x$ is obtained. In the new iteration of recursive process, G_B is set to be $G_B - C_x$. The tip of C_x is considered as a PI vertex of the new G_B. The vertex d is the leading vertex in the new G_B. The saturated floor cone C_c including the set of the vertices $\{c, a, y\}$ is a prime cone and is selected to be in the mapping solution. Again set G_B to be $G_B - C_c$. The subgraph induced by the set of vertices $\{b, d\}$ is selected to be in the mapping solution. It is seen that the mapping solution is an optimal one including three cones.

If the leading vertex of a prime cone is not critical, the selection of the prime cone may lead to a nonoptimal solution. For example, in Figure 9 the cone C_y including the vertex y is a prime cone but its leading vertex is not critical. If we select C_y to be in the mapping solution, eventually three cones, $\{y\}$, $\{c, a, m, n, o, p, x\}$, and $\{d, b\}$, will be selected in the mapping solution. However, the cones $\{a, b, c, d, y\}$ and $\{m, n, o, p, x\}$ can be selected in the mapping solution which is the optimal solution. Hence, the selection of a prime cone whose fan-out vertex is not a critical leading vertex can lead to a nonoptimal solution. Nevertheless, it is difficult to determine whether a prime cone whose fan-out vertex is not a critical leading vertex is selected or not. Its time complexity can be exponential. In practice, mostly, the leading vertex of a prime cone is critical in our experience. Even more, good mapping solutions can be obtained in the

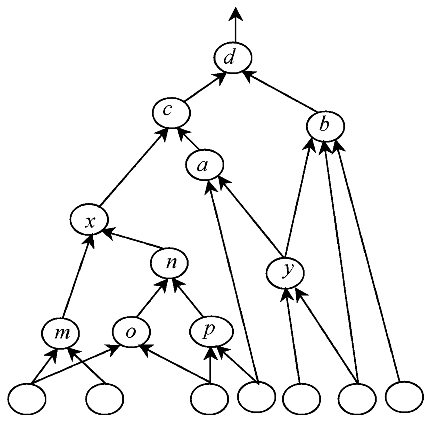


Fig. 9. A feasible cone covers the leading vertex and its all non-PI predecessors ($k = 5$).

experimental results shown in Section 7 if there exist no critical cones and we select a prime cone to be in the mapping solution.

Based on the above discussion, we need an algorithm to find the critical floor cones and the prime cone. To find those cones, we must find the saturated floor cones that can be generated by inspecting all k-feasible floor cones.

It is easy to find all k-feasible floor cones. Let $Pre(v)$ denote the non-PI predecessors of v and $Fin(v)$ denote the set of fan-in vertices of v. Let $SP(v) = \cup_{u \in Fin(v)} Pre(u)$. Clearly, $Pre(v) = \{v\} \cup SP(v)$. Accordingly, starting from the tip of a primary block G_B, all k-feasible floor cones can be found by traversing G_B in postorder. Assume that C_v is a floor cone induced by $Pre(v)$. Recall that C_v is k-feasible if and only if $|Input(C_v)| \le k$ and $|Output(C_v)| = 1$. The procedure to find the k-feasible floor cones in G_B is as follows:

```
Algorithm Generating Floor Cones:
  GenerateFloorCones ($G_B$)
  Comment: $G_B = (V, E)$ is a DAG
  begin
  $stack \leftarrow \oslash$;
  for every vertex $v$ in $V$ do begin
      $timesVisited(v) \leftarrow 0$
      if ($v$ is a PI vertex) then push $v$ onto $stack$
  end of for-loop
  while stack $\neq \oslash$ do begin
      $vx \leftarrow$ pop($stack$);
      $Pre(v) = \{v\} \cup SP(v)$;
      Construct $C_{vx}$ which is the induced subgraph by $Pre(v)$;
      if ( $|Input(C_{vx})| \le k$ and $|Output(C_{vx})| = 1$) then
         add $C_{vx}$ to the list of all $k$-feasible floor cones
  for every fan-out vertex of $vx$, $vs$, do begin
  $timesVisited(vs) \leftarrow timesVisited(vs) + 1$;
  if ($timesVisited(vs) =$ indegree of $vs$) do
     push $vs$ onto stack;
  end of for-loop
     end of while-loop
     end of GenerateAllFloorCones
```

In conclusion, the area-mapping algorithm can be described as follows:

```
Area-minimum Mapping ($G$)
Comment: $G$ is a DAG
begin
  PrimaryBlock_Partitioning($G$);
  for every primary block $G_B$ do begin
    PrimaryBlock_Mapping ($G'_B$);
  end of for-loop
  return the union of the mapping solution of primary blocks;
end of Area-minimum Mapping

PrimaryBlock_Mapping($G'_B$)
  Comment: $G'_B$ is a primary block
  begin
    GenerateFloorCones ($G'_B$);
    Identify the critical floor cones;
    if (there is a critical floor cone, $C_k$) then begin
      add $C_k$ to the Mapping_Solution;
      if ($G'_B - C_k \neq \oslash$) then PrimaryBlock_Mapping ($G'_B - C_k$);
    end
    else bigin
      find the prime cone $C_p$;
      add $C_p$ to the Mapping_Solution;
      if ($G'_B - C_p \neq \oslash$) then PrimaryBlock_Mapping($G'_B - C_p$);
    end
end of PrimaryBlock_Mapping
```

THEOREM 8. *The complexity of generating all k-feasible floor cones in G_B is $O(n^2)$, where n is the total number of vertices.*

PROOF. In the procedure, every edge is traversed once. For every vertex v, we must check whether the union of $\{v\}$ and $SP(v)$ is a feasible cone or not. Therefore the complexity of generating all k-feasible floor cones is bounded by $O(n^2)$. □

THEOREM 9. *Given a k-bounded network, the time complexity of the proposed mapping algorithm is $O(n^3)$, where n is the number of vertices.*

PROOF. The PrimaryBlock_Mapping algorithm must be executed recursively to find the cone of a mapping solution in a given primary block. Every iteration in the recursion finds one cone. The maximum depth of recursion is less than or equal to n. The time complexity of finding a cone is $O(n^2)$. Therefore, for a primary block, the upper bound of the complexity of the greedy method is $O(n \times n^2)$. The time complexity of partitioning the graph into primary blocks is $O(e)$, where e is the total number of edges. The total number of edges is less than n^2. Hence, the total time complexity of the area-optimal mapping algorithm is $O(n^3)$. □

6. EXPERIMENTAL RESULTS

The proposed algorithm was implemented in C language on an Ultra 30 workstation. Testing was accomplished by having the algorithm experimentally design several circuits from the Microelectronics Center of North Carolina

Table I. Comparison of the CPU Runtimes (T, seconds), Number of LUTs (A), and Depths (D) with the DFmap ($k = 5$)

Circuit	Size	Proposed algorithm			DFmap		
		T	A	D	T	A	D
9symml	151	0.19	73	9	12.2	69	7
C3540	956	0.20	425	18	5.5	399	21
alu2	275	0.05	125	14	1.2	120	16
alu4	540	0.08	230	14	1.8	220	18
apex6	699	0.06	229	14	2.0	220	18
C880	302	0.03	117	9	0.5	117	12
rot	436	0.03	202	9	0.6	201	12
i7	340	0.03	146	4	0.6	146	4
C499	370	0.02	66	5	1.0	66	5
duke2	181	0.03	131	8	0.3	129	9
rd84	108	0.02	46	7	0.3	44	7
C5315	1454	16.89	473	12	26.0	448	11
C6288	2353	0.3	1425	90	0.3	1425	90
C7552	2118	57.45	676	11	71.7	645	10
s1196	481	0.03	204	12	0.8	200	12
s1494	558	0.09	230	7	3.4	219	6
s38417	8623	2.85	2869	13	33.2	2708	12
s5378	1074	0.19	483	8	5.2	473	8
des	2244	0.22	1081	10	4.2	1068	10
Total		78.76	9231	274	170.8	9111	288
Comparison		1	1	1	+117%	−1%	+5%

(MCNC) logic synthesis benchmark set. Prior to use of these algorithms, a SIS [Sentovich et al. 1992] environment was used to reduce the complexity of the given network and produce a 2-bounded general Boolean network. Table I presents the comparison of runtimes, the number of LUTs, and depths with the DFmap algorithm [Cong and Ding 1994b]. It is seen that the proposed algorithm tools 117% less CPU time than the DFmap on average. The number of LUTs generated by the proposed algorithm was close to the optimal area-mapping solution found by the DFmap and reduced by 5% the depth compared with the DFmap. Furthermore, for special circuits such as C880, i7, and C499, the number of LUTs generated by the proposed algorithm was the same as the DFmap but took 16–50 less time than the DFmap. In this experiment, k was set to be 5. If k is larger than 5, the DFmap can take much more time than the proposed algorithm.

We carried out the existing postprocessing procedures [Cong and Ding 1994a] to find the gates which can be duplicated such that the total number of LUTs is further reduced. Compared to the existing algorithm [Cong and Ding 1994a; Francis et al. 1991; Cong and Hwang 1995] with gate duplication, the proposed algorithm reduced by 55%, 0.2%, and 21% the number of LUTs compared with the Chortle-crf, the CutMap, and the FlowMap, respectively, as shown in Table II. Therefore, it is possible to generate a solution without duplication first, and then find the part that can be duplicated to reduce the total number of LUTs.

Among the algorithms in Table II, our algorithm and the Chortle-crf are aimed primarily at a minimum number of LUTs, while CutMap and FlowMap

Table II. Comparison of the Number of LUTs (A) and Depths (D) with Chortle-crf, CutMap, and FlowMap ($k = 5$)

Mapper / Circuit	Proposed algorithm		Chortle-crf		CutMap		FlowMap	
	A	D	A	D	A	D	A	D
9symml	59	7	60	9	70	5	77	5
C3540	311	17	319	16	460	10	549	10
alu2	110	20	111	19	137	9	171	9
alu4	197	21	196	21	254	10	300	10
apex6	196	14	193	18	255	10	300	10
C880	84	11	94	14	143	7	161	7
rot	191	12	203	14	230	7	266	7
i7	103	2	107	2	107	2	139	2
C499	66	5	70	6	66	4	74	4
duke2	126	8	116	8	159	4	190	4
rd84	35	5	40	7	36	4	44	4
C5315	428	9	439	12	571	8	690	8
C6288	524	29	494	29	646	22	760	22
C7552	624	8	644	10	653	7	741	7
s1196	161	11	166	10	184	5	211	5
s1494	198	6	188	8	229	4	260	4
s38417	2504	11	2555	9	2917	8	4134	8
s5378	421	9	420	9	463	5	527	5
des	1010	10	950	11	1096	5	1552	5
Total	7348	215	7365	232	8886	155	11,398	155
Comparison	1	1	+0.2%	+8%	+21%	−28%	+55%	−28%

focus on minimizing the depth of the mapping solution. The experimental results show that CutMap and FlowMap generate better solutions than our algorithm and Chortle-crf in depth. These comparisons show that a good mapping algorithm should first consider the primary objective and then preferably allow controllable tradeoff among all objectives. Otherwise, the mapping results will be of low quality for the primary objective.

The effect of the divide-and-conquer algorithm used to partition the given graph into primary blocks is shown in Table III. It is seen that after executing the partitioning algorithm, the area-optimal duplication-free mapping solution was found in circuit C6288, which is an array multiplier. Hence, the percentage of the C6288 was 100%. This is the reason that the CPU runtime of the C6288 was much faster than the other approximate circuits such as C7552. For circuit 9symml, this divide-and-conquer step generated zero feasible blocks such that the DFmap had to use the dynamic programming method to find the solution. It needed a large amount of CPU runtime, as shown in Table I. Obviously, the primary block partitioning algorithm can speed up the performance of the proposed algorithm.

To demonstrate the effectiveness of our algorithm, Table IV presents the number of critical floor cones and prime cones of every benchmark. According to the discussions in Section 5, a selection of a prime cone whose fan-out vertex is not a critical leading vertex can lead to a nonoptimal solution. In Table IV, we find that the number of the prime cones whose leading vertex is noncritical did

Table III. Results of the Primary Block Partitioning Algorithm ($k = 5$)

Item / Circuit	# Primary blocks	# Feasible blocks found by the proposed algorithm (L_N)	# Feasible blocks which are also feasible blocks (B_N)	Ratio ($\frac{B_N}{L_N}$)
9symml	1	73	0	0%
C3540	235	425	190	45%
alu2	54	125	44	35%
alu4	104	230	84	37%
apex6	104	229	84	37%
C880	65	117	43	37%
rot	144	202	119	59%
i7	80	146	14	10%
C499	58	66	50	76%
duke2	94	131	79	60%
rd84	22	46	18	39%
C5315	263	473	176	37%
C6288	1425	1425	1425	100%
C7552	383	676	326	48%
s1196	123	204	98	48%
s1494	56	230	31	13%
s38417	2014	2869	1612	56%
s5378	318	483	263	54%
des	587	1081	399	37%

not exceed 6% of the total LUTs number. Even more, circuits C880, i7, and C499 generated optimal solutions. Therefore, we believe that good mapping solutions can be obtained by using our algorithm.

7. CONCLUSIONS

The increasing popularity of LUT-based FPGAs technology and the unique features of the architecture have led to intensive studies on design automation techniques for LUT-based FPGAs. This article proposed an area-minimum algorithm without gate duplication for the technology mapping problem of LUT-based FPGAs. By combining the proposed algorithm with the two existing post-processing procedures [Cong and Ding 1994a] which are used to find that the gates can be duplicated, the total number of LUTs is reduced.

The main contribution of this article has focused on finding an algorithm which approximates to the optimal solution and has lower time complexity. The time complexity of this algorithm was proven theoretically to be bounded by $O(n^3)$, where n is the total number of gates in the given circuit. Although DFmap [Cong and Ding 1994b] found an optimal solution in polynomial time for the minimal-area FPGA technology mapping problem without gate duplication, a major shortcoming in the approach is that its time complexity can grow exponentially with the number of inputs of the LUTs. Our experis mental results showed that DFmap can take a large amount of runtime for some homogeneous FPGAs consisting of LUTs with a very large input size. The proposed algorithm can not only greatly reduce the runtime for homogeneous FPGAs but it can also be used for heterogeneous FPGA because the time complexity of the algorithm is irrelevant to the number of inputs of the LUTs. Our future research will

Table IV. Circuit Optimization Using the Greedy Method ($k = 5$)

Circuit	# Total feasible blocks (L_N)	Critical floor cones		Prime cones whose leading vertex is critical		Prime cones whose leading vertex is non-critical	
		A (C_N)	Ratio ($\frac{C_N}{L_N}$)	A (P_{CN})	Ratio ($\frac{P_{CN}}{L_N}$)	A (P_{NCN})	Ratio ($\frac{P_{NCN}}{L_N}$)
9symml	73	60	82%	9	12%	4	5%
C3540	425	182	43%	27	6%	26	6%
alu2	125	65	52%	11	9%	5	4%
alu4	230	114	50%	22	10%	10	4%
apex6	229	112	49%	24	10%	9	4%
C880	117	57	49%	17	15%	0	0%
rot	202	61	30%	21	10%	1	0%
i7	146	102	70%	30	21%	0	0%
C499	66	8	12%	8	12%	0	0%
duke2	131	42	32%	8	6%	2	2%
rd84	46	24	52%	2	4%	2	4%
C5315	473	257	54%	15	3%	25	5%
C6288	1425	0	0%	0	0%	0	0%
C7552	676	315	47%	4	1%	31	5%
s1196	204	90	44%	12	v6%	4	2%
s1494	230	153	67%	35	15%	11	5%
s38417	2869	873	30%	223	8%	161	6%
s5378	483	178	37%	32	7%	10	2%
des	1081	497	46%	172	16%	13	1%

include two parts: (1) finding more efficient algorithms than the existing methods which are used to find that the gates can be duplicated to further minimize the number of LUTs and (2) finding an area-minimum mapping solution for the heterogeneous FPGA by using the proposed algorithm.

REFERENCES

Chowdhary, A. and Hayes, J. P. 1995. Technology mapping for field programmable gate arrays using integer programming. In *Proceedings of the IEEE International Conference on Computer-Aided Design*. 564–567.

Chen, G. and Cong, J. 2001. Simultaneous logic decomposition with technology mapping in FPGA designs. In *Proceedings of the ACM/SIGDA International Symposium on Field Programmable Gate Arrays*. 48–55.

Cong, J. and Ding, Y. 1994a. FlowMap: An optimal technology mapping algorithm for delay optimization in lookup-table based FPGA designs. *IEEE Trans. Comput.-Aided Des. of Integr. Circ. Syst. 13*, 1 (Jan.), 1–11.

Cong, J. and Ding, Y. 1994b. On area/depth trade-off in LUT-Based FPGA technology mapping. *IEEE Trans. VLSI Syst. 2*, 2 (June), 137–148.

Cong, J., Ding, Y., Kahug, A., and Trajmar, P. 1992. An improved graph-based FPGA technology mapping algorithm for delay optimization. In *Proceedings of the IEEE International Conference on Computer Design*. 154–158.

Cong, J. and Hwang, Y. Y. 1995. Simultaneous depth and area minimization in LUT-based FPGA mapping. In *Proceedings of the ACM International Symposium on Field Programmable Gate Arrays*. 68–74.

Cong, J., Wu, C., and Ding, E. 1999. Cut ranking and pruning: Enabling a general and efficient FPGA mapping solution. In *Proceedings of the ACM International Symposium on Field Programmable Gate Arrays*. 29–35.

Farrahi, H. and Sarrafzadeh, M. 1994. Complexity of the lookup-table minimization problem for FPGA technology mapping. *IEEE Trans. Comput.-Aided Des. Integr. Circ. Syst. 13*, 11(Nov.), 1319–1332.

FRANCIS, J., ROSE, J., AND CHUNGM, M. K. 1990. Chortle: A technology mapping program for lookup table-based field programmable gate arrays. In *Proceedings of the Design Automation Conference*. 613–619.

FRANCIS, J., ROSE, J., AND VRANESIC, Z. 1991. Chortle-crf: Fast technology mapping for lookup table-based FPGAs. In *Proceedings of the Design Automation Conference*. 248–251.

HUANG, J. D., JOU, J. Y., AND SHEN, W. Z. 1995. Compatible class encoding in Roth-Karp decomposition for two-output LUT architecture. In *Proceedings of the IEEE International Conference on Computer-Aided Design*. 359–363.

HWANG, T. T., OWENS, R. M., AND IRWIN, M. J. 1994. Logic synthesis for field-programmable gate arrays. *IEEE Trans. Comput.-Aided Des. 13*, 10(Oct.), 1280–1287.

KARP, R. M. AND ROTH, J. P. 1962. Minimization over Boolean graphs. *IBM J. Res. Develop.* April, 227–238.

KARPLUS, K. 1991. Xmap: A technology mapper for table-lookup field-programmable gate arrays. In *Proceedings of the Design Automation Conference*. 240–243.

LAI, Y. T., PAN, K. R., AND PEDRAM, M. 1994. FPGA synthesis using function decomposition. In *Proceedings of the IEEE International Conference on Computer-Aided Design.* 30–35.

LAI, Y. T., PEDRAM, M., AND VRUDHULA, S. B. K. 1993. BDD based decomposition of logic functions with application to FPGA synthesis. In *Proceedings of the Design Automation Conference*. 642–647.

LEHMAN, E., WATANABE, Y., GRODSTEIN, J., AND HARKNESS, H. 1997. Logic decomposition during technology Mapping. *IEEE Trans. Comput.-Aided Des. Integr. Circ. Syst. 16*, 8(Aug.), 813–834.

MURGAI, R., NISHIZAKI, Y., SHENOY, N., BRAYTON, R. K., AND SANGIOVANNI-VINCENTELLI, A. 1990. Logic synthesis algorithms for programmable gate arrays. In *Proceedings of the Design Automation Conference*. 620–625.

MURGAI, R., SHENOY, N., BRAYTON, R. K., AND SANGIOVANNI-VINCENTELLI, A. 1991. Improved logic synthesis algorithms for table look up architectures. In *Proceedings of the IEEE International Conference on Computer-Aided Design*. 564–567.

SAWADA, H., SUYAMA, T., AND NAGOYA, A. 1995. Logic synthesis for look-up table based FPGA's using functional decomposition and support minimization. In *Proceedings of the IEEE International Conference on Computer-Aided Design*. 353–358.

SAWKAR, P. AND THOMAS, D. 1992. Area and delay mapping for table-look-up based field programmable gate arrays. In *Proceedings of the Design Automation Conference*. 368–373.

SENTOVICH, E., SINGH, K., LAVAGNO, L., MOON, C., MURGAI, R., SALDANHA, A., SAVOJ, H., STEPHAN, P., BRAYTON, R., AND SANGIOVANNI-VINCENTELLI, A. 1992. SIS: A system for sequential circuit synthesis. Tech. rep. UCB/ERL M92/41. University of California, Berkeley, Berkeley, CA.

SHEN, W. Z., HUANG, J. D., AND CHAO, S. M. 1995. Lambda set selection in Roth-Karp decomposition for LUT-based FPGA technology mapping. In *Proceedings of the Design Automation Conference*. 65–69.

WOO, N. S. 1991. A heuristic method for FPGA technology mapping based on the edge visibility. In *Proceedings of the Design Automation Conference*. 248–251.

WURTH, B., ECKL, K., AND ANTREICH, K. 1995. Functional multiple-output decomposition: Theory and an implicit algorithm. In *Proceedings of the Design Automation Conference*. 54–59.

ZHANG, S., MILLER, D. M., AND MUZIO, J. C. 1996. Notes on complexity of the lookup-table minimization problem for FPGA technology mapping. *IEEE Trans. Comput.-Aided Des. Integr. Circ. Syst. 15*, 12(Dec.), 1588–1590.

Received January 2003; revised November 2003, April 2004; accepted August 2004