On Circuit Clustering for Area/Delay Tradeoff Under Capacity and Pin Constraints

Juinn-Dar Huang, Jing-Yang Jou, Wen-Zen Shen, and Hsien-Ho Chuang

***Abstract*— In this paper, we propose an iterative area/delay tradeoff algorithm to solve the circuit clustering problem under the capacity constraint. It first finds an initial delay-considered area-optimized clustering solution by a delay-oriented depth-first-search procedure. Then, an iterative procedure consisting of several reclustering techniques is applied to gradually trade the area for the performance. We then show that this algorithm can be easily extended to solve the clustering problem subject to both capacity and pin constraints. Experimental results show that our algorithm can provide a complete set of clustering solutions from the area-optimized one to the delay-optimized one for a given circuit. Furthermore, comparing to the existing delay-optimized algorithms, ours achieves almost the same performance but with much less area overhead. Therefore, this algorithm is very useful on solving the timing-driven circuit clustering problem.**

***Index Terms*— Clustering, critical path, delay, partitioning, performance, performance tradeoffs.**

I. Introduction

PARTITIONING techniques play important roles in many aspects of modern VLSI circuit designs. Many partitioning problems on all levels of abstraction have been studied for several decades [1]. In general, a number of design constraints such as capacity constraints, pin constraints and timing constraints, must be met during the partitioning process. For example, if a circuit cannot be fit into one chip, it must be divided into several subcircuits such that each of them can be put into a chip. However, an improper partitioning could dramatically degrade the performance of the original circuit. Therefore, partitioning a large circuit into chips to minimize the circuit delay under the capacity constraint of a chip is a very important problem.

This problem has been studied in several previous works [2]–[4]. These works all involve the replication of logic gates, that is, a logic gate can be assigned to more than one block. Such a block is referred to as a *cluster*, and the circuit partitioning problem is referred to as the *circuit clustering problem*. The *LLT algorithm*, presented by Lawler, Levitt, and Turner, is a polynomial time algorithm which can obtain an optimal solution of this problem under the *unit delay model* [2]. In the unit delay model: 1) all gate delays are zero, 2) no delay is associated with an interconnection linking two gates in the same cluster, and 3) a delay of one time unit is associated with an interconnection linking two gates in two different clusters. However, as more gates are packed into a cluster, a path could consist of many gates within a cluster and could incur a substantial delay within a cluster. Therefore, the unit delay model may not be very realistic because it assumes the total delay incurred within a cluster is negligible. Then, the *generalized LLT algorithm*, referred to as the *GLLT algorithm*, was proposed in [3] to enhance LLT by replacing the unit delay model with a more practical model, the *general delay model*. In this general delay model, we have the following:

1) each gate v has an intrinsic delay of $\delta(v)$;
2) no delay is associated with an interconnection linking two gates in the same cluster;
3) a delay of D time units (D is a specific constant) is associated with an interconnection linking two gates in two different clusters.

GLLT is a heuristic algorithm which uses a greedy labeling approach to minimize the circuit delay without replicating many gates. Recently, Rajaraman and Wong proposed another algorithm which can get the delay optimal solution for any combinational circuit under the general delay model with polynomial time complexity [4]. However, the number of clusters can be four times as high as that of the area-optimized solution on average according to our experimental results.

For some applications, we have to pay whatever area overhead (number of clusters) needed to get the minimal delay. However, the normal requirement is to get a solution which can meet the timing constraint while keeping the area overhead as small as possible. In this paper, we present a new approach which can generate a set of clustering solutions from the area-optimized one to the delay-optimized one for a given circuit under the general delay model subject to the capacity constraint. It starts from getting an area-optimized initial clustering solution with delay consideration. Then, a series of reclustering operations is applied to gradually decrease the delay without increasing too many extra clusters. Experimental results show that this approach can effectively generate a set of comprehensive clustering solutions with different area/delay tradeoffs for a given circuit. Moreover, the delay-optimized solutions obtained by our algorithm are identical or very close (within 1.7% on average) to the delay optimal solutions obtained by [4] with much lower area overhead as compared to them. Moreover, this algorithm can be easily extended to solve the clustering problem subject to both capacity and pin constraints.

Additionally, a clustering technique with a different objective is proposed [5]. It logically clusters a large circuit into

Manuscript received August 17, 1997; revised December 22, 1997 and March 31, 1998. This work was supported by the ROC National Science Council under Grant NSC 87-2215-E-009-040.

The authors are with the Department of Electronics Engineering, National Chiao Tung University, Hsinchu, Taiwan.

Publisher Item Identifier S 1063-8210(98)05989-7.

several small circuits and then dispatches them to the delay minimizer separately. This strategy can shorten the overall delay minimization process because the complexity of existing delay minimizers is usually worst than linear in terms of the circuit size. Therefore, it is claimed that this technique can speed up the delay minimization process without losing the overall optimization quality significantly. However, the difference between this technique and previously described works is that the target circuit is only clustered for efficient delay minimization temporarily and those small clusters will be eventually reglued to be a single delay-minimized circuit.

The rest of this paper is organized as follows. Section II introduces some basic terminology and definitions used in this paper. Section III describes the way to get the delay-considered area-optimized initial clustering solution for a given circuit. In Section IV, our reclustering techniques for performing area/delay tradeoff are presented in detail. Section V shows the comprehensive experimental results. The extension targeting the pin constraint is discussed in Section VI. The concluding remarks are given in Section VII.

II. Preliminaries

A combinational Boolean network N can be represented as a directed acyclic graph, $G(V, E)$. Each node, $v \in V$, represents a logic gate and each directed edge, $\langle i, j\rangle \in E$, i and $j \in V$, represents that node i is a fan-in of node j. Let input (v) be the set consisting of all fan-ins of v, and output (v) be the set consisting of all fan-outs of v. A *primary input* (*PI*) of a network is a node without any incoming edge and a *primary output* (*PO*) of a network is a node without any outgoing edge. Each node v has an intrinsic delay $\delta(v)$ associated with it. Each node v also has a weight $w(v)$ associated with it to represent the corresponding area cost. The weight of a set of nodes S, denoted as $W(S)$, is the sum of $w(v)$ where $v \in S$. A *cluster* C is a set of nodes $\in V$ and the *capacity constraint* of C specifies that $W(C)$ must not exceed a constant M. The *label* of a node v, denoted as $l(v)$, represents the maximum delay along any path starting from PI's and ending at v. The label of each PI is defined as 0. Then, under the general delay model, the label of any other node v can be calculated as

$$l(v) = \max_{u \in \mathrm{input}(v)} l(u) + \delta(v)$$

where

$$l(u) = \begin{cases} l(u), & \text{if edge } \langle u, v\rangle \text{ is internal to a cluster,} \\ l(u) + D, & \text{if edge } \langle u, v\rangle \text{ crosses a cluster boundry.} \end{cases}$$

The label of a network N, denoted as $l(N)$, is defined as the maximum label of PO's. The *required label* of a network N, which is specified by the users and denoted as $rl(N)$, indicates the allowed maximum delay of the resultant clustered network. Thus, for each PO v, the required label, denoted as $rl(v)$, is implicitly set to $rl(N)$ by definition. Then, the required label of any other node v, can be calculated as

$$rl(v) = \min_{u \in \mathrm{output}(v)} rl'(u) - \delta(u)$$

```
Initial_Acyclic_Partitioning(network Net, capacity M) {
    L ← Topological_Sorting(Net);
    S ← ∅;
    A ← Allocate a new block;
    Insert_Last(S, A);        /* append A at the end of S */
    while(L is not empty) {
        Node ← Pop_First(L);
        if(W(A) + W(Node) > M) {     /* violate the capacity constraint ? */
            A ← Allocate a new block;
            Insert_Last(S, A);
        }
        A ← A ∪ {Node};
    }
    return S;     /* S is an acyclic partitioning solution */
}
```

Fig. 1. The initial acyclic partitioning algorithm.

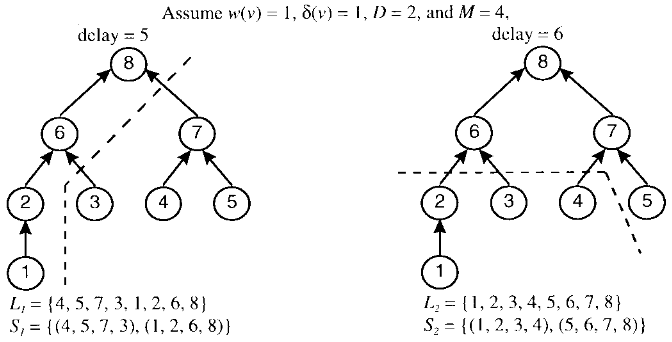
Assume w(v) = 1, δ(v) = 1, D = 2, and M = 4,
delay = 5
delay = 6
8
6
7
2
3
4
5
1
L_1 = {4, 5, 7, 3, 1, 2, 6, 8}
S_1 = {(4, 5, 7, 3), (1, 2, 6, 8)}
L_2 = {1, 2, 3, 4, 5, 6, 7, 8}
S_2 = {(1, 2, 3, 4), (5, 6, 7, 8)}

Fig. 2. Acyclic partitioning solutions of two valid topological sorting lists.

where

$$rl(u) = \begin{cases} rl(u) & \text{if edge } \langle v, u\rangle \text{ is internal to a cluster,} \\ rl(u) - D & \text{if edge } \langle v, u\rangle \text{ crosses a cluster boundry.} \end{cases}$$

A node v is a *failing node* if $rl(v)$ is smaller than $l(v)$. A *failing path* is a path starting from a PI and ending at a PO and contains all failing nodes because this path fails to achieve the required circuit delay.

III. The Initial Clustering Algorithm

Given a partitioning solution $S = \{A_1, A_2, \cdots, A_n\}$, authors of [6] define the *dependency graph* of S, denoted as $D(S)$, such that there exists an edge $\langle A_i, A_j\rangle$ if and only if there exists an edge $\langle x, y\rangle$ in the network where node $x \in A_i$ and node $y \in A_j$. S is called an *acyclic partitioning solution* if $D(S)$ is a directed acyclic graph. Some benefits can be obtained if the partitioning solution is acyclic [6]. It is noticeable that the clustering solutions provided by [4] are acyclic though the algorithm does not explicitly try to maintain this property.

Our algorithm starts from finding an area-optimized clustering solution with delay consideration where the area is counted as the number of clusters. The problem of packing nodes with different weights into minimum number of clusters with the specified capacity constraint can be formulated as the *bin packing problem* in which nodes are treated as boxes with different sizes and clusters are considered as bins with the fixed capacity. Therefore, in this paper an effective heuristic

```
Topological_Sorting(network Net) {
    Label each node $v$ as the largest intrinsic delay between PIs and the node $v$;
    $S \leftarrow \emptyset$;
    PO_List ← Sort POs in decreasing order of their labels;
    while(PO_List is not empty) {
        Node ← Pop_First(PO_List);
        DFS(Node, S);
    }
    return S;
}
DFS(node Node, list S) {
    Fanin_List ← Sort all unmarked fanins of Node in increasing order of their labels;
    foreach(Fanin ∈ Fanin_List)
        DFS(Fanin, S);
    mark Node;
    Insert_Last(S, Node);    /* Append Node at the end of S */
}
```

Fig. 3. Our delay-oriented topological sorting list generation algorithm.

algorithm based on the notion of the acyclic partitioning is used to get our initial clustering solutions. In order to minimize the number of clusters, node replication is not allowed in getting the initial solution. The algorithm is shown in Fig. 1. At first, a topological sorting is performed over the nodes of the network to get a sorted list L. Node i appears before node j in L if node i is a fan-in of node j. Then, as many nodes from the beginning of L are packed into a block as long as the capacity constraint is not violated. A new block is allocated when the current block is fully filled under the capacity constraint. This process is not terminated until all nodes are packed. It is obvious that the resulting partitioning solution S is acyclic.

However, there are many ways in performing topological sorting. Clearly, different sorted lists may lead to different partitioning solutions, and then result in different circuit delays. For instance, consider the example illustrated in Fig. 2. L_1 and L_2 are two different topologically sorted lists and S_1 and S_2 are their corresponding acyclic partitioning solutions. Both solutions consist of two blocks. However, S_1 is better than S_2 in terms of circuit delay. The reason why S_1 has the smaller delay is because it puts the longest path entirely into the same block. Since the delay of a path is the sum of both intrinsic and intercluster delays, the only way to reduce the circuit delay is to minimize intercluster delays for those paths with large intrinsic delays. Therefore, nodes of such a path should be packed within as few clusters as possible. This implies that nodes of such a path should be put as close as possible during topological sorting. Following this principle, we propose a delay-oriented topological sorting algorithm shown in Fig. 3.

This algorithm first labels each node of the network by the largest intrinsic delay between PI's and this node. A topologically sorted list L can then be obtained by recursively applying the depth-first-search (DFS) starting from PO's to PI's. Starting at the PO with the largest label, DFS is applied to unmarked fan-ins of any node v in increasing order of their labels while v is visited. Here we exploit a useful property of the stack while DFS is called recursively. Since nodes with smaller labels are visited first, DFS calls applied to nodes with larger labels tend to stay in the stack during the recursion. Therefore, those DFS calls applied to nodes on the longest path tend to concurrently terminate at the end of a series of DFS calls and the corresponding nodes are more likely inserted into the sorted list S at the same time. As a result, the intercluster delay can be reduced. This technique is extremely effective for tree-intensive circuits as illustrated in L_1 of Fig. 2. In this algorithm, DFS is applied to PO's in decreasing order of their labels. It is because the path ending at the PO with the largest label should be considered first.

Our initial clustering algorithm, referred to as *INITIAL*, is implemented in SIS environment [8] and is compared to the algorithm proposed in [4], referred to as *CLUSTER-RW*.[1] Table I shows the results of comparing *CLUSTER-RW* against *INITIAL* under the given parameters, $\delta(v) = 1, w(v) = 1, D = 2$, and $M = 100$. The number of clusters obtained by *INITIAL* is minimal, i.e., $\lceil \#node/M \rceil$, because w is set to 1 and no logic synthesis is allowed. Also, because δ is identical for all nodes, the maximum intrinsic delay of the circuit is calculated as δ times its level lvl where lvl represents the maximum number of nodes from PI's to PO's along any path. Thus, the upper bound of the circuit delay, shown in Column *U.B.*, can be calculated as $\delta \times lvl + D \times (\min(lvl, \#cluster) - 1)$. This upper bound gives us the worst delay a circuit can have if an improper partitioning is applied. Of course, the lower bound of the circuit delay must be the one obtained from *CLUSTER-RW*. From Table I, the number of clusters produced by *CLUSTER-RW* is three times more than that produced by *INITIAL* on average. The circuit delay obtained by *INITIAL* is 23% smaller than the upper bound while the optimal delay is 18% smaller than that of *INITIAL*. The results show that *INITIAL* can generate clustering solutions with delays closer

[1]There is no experimental data associated with the number of nodes and clusters in [4]. We get this data by running the software package provided by the authors. Since *CLUSTER-RW* does not guarantee to get the minimum number of clusters in the optimal delay solutions, the postprocessing techniques developed by authors' successive work [7] are used to reduce the number of clusters without losing the delay optimality.

TABLE I
CLUSTER-RW VERSUS *INITIAL*, UNDER $\delta(v) = 1$, $w(v) = 1$, $D = 2$, $M = 100$

	circuit information			*CLUSTER-RW*			*INITIAL*	
circuit	#node	level	U.B.	delay	#node	#cluster	delay	#cluster
c499	202	11	15	11	3200	32	15	3
c880	357	23	29	23	848	9	25	4
c1355	514	23	33	25	3560	36	29	6
c1908	880	40	56	41	4272	44	47	9
c2670	1161	32	54	34	2073	21	42	12
c3540	1667	47	79	48	7496	76	61	17
c5315	2290	49	93	51	8230	83	58	23
c6288	2416	124	172	128	13647	137	166	25
c7552	3466	43	111	44	9786	98	50	35
total	12953	392	642	405	53112	536	493	134

```
Iterative_Area/Delay_Trade-Off(network Net, delay Target_Delay) {
    Original_Delay ← l(Net);
    Work_Net ← Copy(Net);        /* including clustering assignment */
    while(l(Work_Net) > Target_Delay) {
        Failing_Path_Set ← Delay_Analysis(Work_Net, l(Work_Net) − 1);
        Failing_Path ← Choose one path from Failing_Path_Set;
        if(Reclustering(Failing_Path) == Success) {          /* a local success */
            if(l(Work_Net) < l(Net))        /* also a global success */
                Net ← Work_Net;              /* including clustering assignment */
        }
        else /* a local failure, also a global failure */
            break;        /* exit while loop */
    }
    Post_Processing(Net);
    if(l(Net) == Target_Delay)    return Success;
    else if(l(Net) < Original_Delay)    return Partial_Success;
    else    return Failure;
}
```

Fig. 4. Our iterative area/delay tradeoff reclustering algorithm.

to the optimal values without any area overhead. Therefore, *INITIAL* can effectively provide a good delay-considered area-optimized initial clustering solution for a given circuit.

IV. THE ITERATIVE AREA/DELAY TRADEOFF RECLUSTERING ALGORITHM

Given an initial delay-considered area-optimized clustering solution, the goal of our algorithm is to reduce the circuit delay without increasing too many extra clusters. The outline of this algorithm is shown in Fig. 4. Given a network, our algorithm iteratively reduces the circuit delay until the desired target delay is achieved or there is no improvement can be made. In our algorithm, tradeoff operations are performed iteratively while the current circuit delay is still larger than the target delay. At each iteration, the required label of the network is set to the current label minus "1." The delay analysis is followed to identify all failing paths in the network. Then, a failing path is selected as the candidate for the delay optimization. As mentioned before, the intrinsic delay of a path cannot be reduced because no logic resynthesis is performed. So the only way to reduce the delay of a failing path is to reduce the intercluster delay. In this paper, we propose three techniques based on node movement and node replication to achieve this goal. These techniques are referred to as *reclustering* techniques. The major feature of them is that they can eliminate the target failing path without introducing extra new ones. If reclustering techniques can eliminate the given failing path, a *local success* is marked. If a local success results in a global success, i.e., the delay of the current working network *Work_Net* is smaller than that of the original network *Net*, then the original network is updated. After a local success, the whole procedure starting from the delay analysis is repeated because the network has been modified. This process is repeated until the target delay is achieved, or no improvement is possible. At last, a postprocessing step is performed to further reduce the number of required clusters without sacrificing the circuit delay.

Given an acyclic initial clustering solution S, each cluster $C \in S$ can be numbered in its topological order, and the number is denoted as $ID(C)$. Therefore, $node_1 \in C_1$ can be a fan-in of $node_2 \in C_2$ only if $ID(C_1) \leq ID(C_2)$. A path

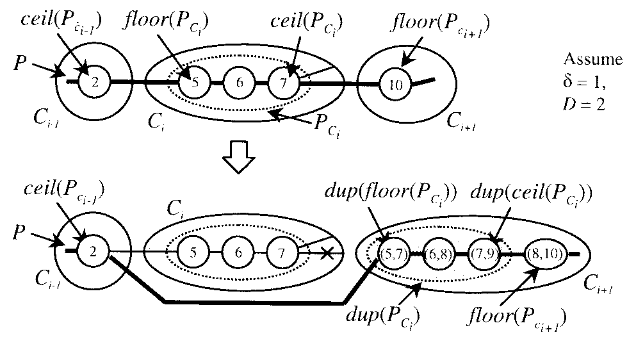

Fig. 5. Illustration of the reclustering technique 1.

P crossing k clusters forms an ordered list $\{C_1, C_2, \cdots, C_k\}$ in increasing order of these clusters' ID's. All PI's/PO's are clustered in C_0/C_{k+1} to simplify the later derivations. We define P_C as the subpath of P residing on the cluster C, $floor(P_C)$ as the first node of P_C, and $ceil(P_C)$ as the last node of P_C. $\Delta(u, v)$ represents the maximum intrinsic delay from node u to node v; $dup(a)$ is a duplication of a where a can be a node or a set of nodes; and $cone_C^P$ is a set of nodes consisting of $ceil(P_C)$ and some of its predecessors $v \in C$ where

$$l(v) + D + \Delta(v, ceil(P_C)) \geq l(ceil(P_C)). \tag{1}$$

These terms are also clearly illustrated in Figs. 5 and 6. The definition of $cone_C^P$ leads to an important property.

Property 1: $l(dup(ceil(P_C)))$ is identical to $l(ceil(P_C))$ if $cone_C^P$ is entirely duplicated to another cluster.

Proof: Assume $v \in cone_C^P$, u is a fan-in of v, $u \notin cone_C^P$, and $u \in C$. After duplicating $cone_C^P$ to another cluster, some edges $\langle u, dup(v)\rangle$ are introduced to connect u and $dup(v) \in dup(cone_C^P)$. Thus, the label of $dup(v)$ may increase because an extra intercluster delay D is added. Thus, the following inequity holds for $v \in cone_C^P$:

$$l(v) \leq l(dup(v)). \tag{2}$$

However, according to (1), for u is a fan-in of v, $u \notin cone_C^P$ and $u \in C$

$$\begin{aligned} l(u) &+ D + \delta(v) + \Delta(dup(v), dup(ceil(P_C))) \\ &= l(u) + D + \Delta(u, ceil(P_C)) < l(ceil(P_C)). \end{aligned} \tag{3}$$

In other words, even though some edges $\langle u, dup(v)\rangle$ incur an extra delay D, those paths pass through them will not cause $l(dup(ceil(P_C)))$ to be greater than $l(ceil(P_C))$. Therefore, from (2) and (3), $l(dup(ceil(P_C)))$ is identical to $l(ceil(P_C))$. Property 1 implies that $cone_C^P$ should be entirely duplicated if we want $l(dup(ceil(P_C))) = l(ceil(P_C))$ after the duplication. □

Technique 1: Duplicate P_{C_i} to $C_i + 1$ where $1 \leq i \leq k-1$ if the capacity constraint of C_{i+1} is still met. Then, replace the fan-in of $floor(P_{C_{i+1}})$, that is $ceil(P_{C_i})$, with $dup(ceil(P_{C_i}))$ as shown in Fig. 5.

Before applying Technique 1, two equities hold

$$l(floor(P_{C_{i+1}})) = l(ceil(P_{C_i})) + D + \delta(floor(P_{C_{i+1}})) \tag{4}$$

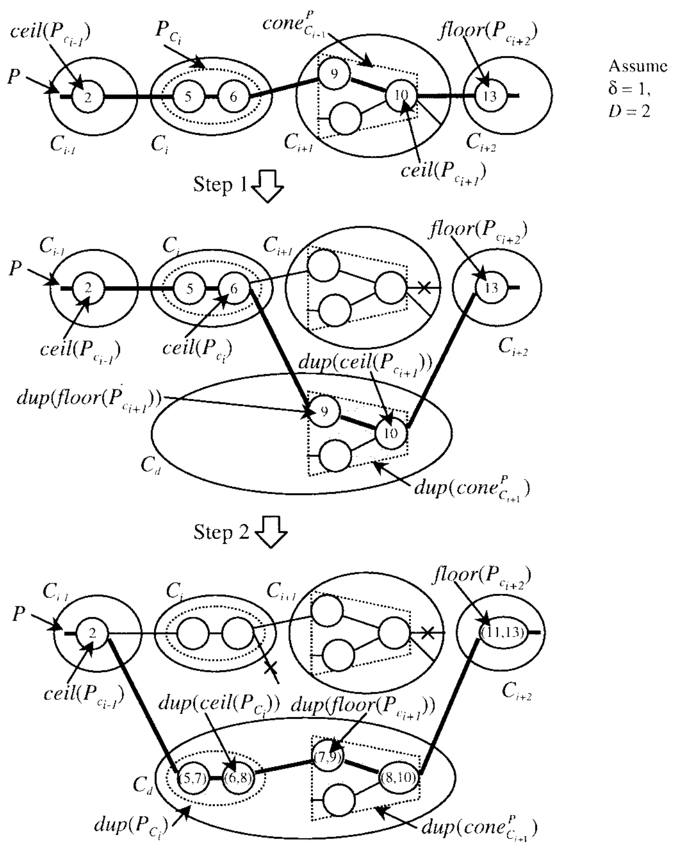

Fig. 6. Illustration of the reclustering technique 2.

$$\begin{aligned} l(ceil(P_{C_i})) &= l(ceil(P_{C_{i-1}})) + D \\ &\quad + \Delta(floor(P_{C_i}), ceil(P_{C_i})). \end{aligned} \tag{5}$$

After applying Technique 1, all paths passing through $\langle ceil(P_{C_i}), floor(P_{C_{i+1}})\rangle$ change their ways by passing through $\langle dup(ceil(P_{C_i})), floor(P_{C_{i+1}})\rangle$. If there exist $v \in P_{C_i}$, u is a fan-in of v, $u \in C_i$ and $u \notin P_{C_i}$, then $l(dup(v))$ may be larger than $l(v)$ because an extra delay D is added on edge $\langle u, dup(v)\rangle$. Thus, the following inequity holds for $v \in P_{C_i}$:

$$l(v) \leq l(dup(v)) \leq l(v) + D. \tag{6}$$

Property 2: The circuit delay never increases after applying Technique 1.

Proof: Let $l_{\text{old}}/l_{\text{new}}$ be the labels of $floor(P_{C_{i+1}})$ before/after the duplication, respectively. Then, according to (6):

$$\begin{aligned} l_{\text{new}} &= l(dup(ceil(P_{C_i}))) + \delta(floor(P_{C_{i+1}})) \\ &\leq l(ceil(P_{C_i})) + D + \delta(floor(P_{C_{i+1}})) = l_{old}. \end{aligned}$$

It means that $l(floor(P_{C_{i+1}}))$ does not increase after applying Technique 1 and neither does the circuit delay. □

Property 3: The target failing path P can be eliminated after applying Technique 1.

Proof: According to (6), this property can be proved in two different cases.

Case 1: When $l(ceil(P_{C_i})) < l(dup(ceil(P_{C_i})))$.

After applying Technique 1, if there exist $v \in P_{C_i}$, u is a fan-in of v, $u \in C_i$ and $u \notin P_{C_i}$, then $l(dup(v))$ may be larger than $l(v)$ because an extra delay D is added on edge $\langle u, dup(v)\rangle$. The effect of $l(dup(v)) > l(v)$ could further propagate toward its fan-outs and eventually causes $l(dup(ceil(P_{C_i}))) > l(ceil(P_{C_i}))$. Thus, according to the assumption of Case 1:

$$\begin{aligned} l(ceil(P_{C_i})) &= l(ceil(P_{C_{i-1}})) \\ &\quad + D + \Delta(ceil(P_{C_{i-1}}) \\ ceil(P_{C_i})) &= l(ceil(P_{C_{i-1}})) \\ &\quad + D + \Delta(dup(ceil(P_{C_{i-1}})) \\ dup(ceil(P_{C_i}))) &< l(dup(ceil(P_{C_i}))). \end{aligned} \tag{7}$$

Equation (7) implies that P is no longer a failing path.

Case 2: When $l(ceil(P_{C_i})) = l(dup(ceil(P_{C_i})))$.

This case means that $l(dup(v))$ does not increase even if an extra delay D is added to such edge $\langle u, dup(v)\rangle$ described in Case 1. Thus, let l_{old}/l_{new} be the labels of $floor(P_{C_{i+1}})$ before/after the duplication, respectively. Then,

$$\begin{aligned} &l(dup(ceil(P_{C_i}))) + \delta(floor(P_{C_{i+1}})) \\ &\quad < l(ceil(P_{C_i})) + D + \delta(floor(P_{C_{i+1}})) = l_{old}. \end{aligned} \tag{8}$$

Because $dup(ceil(P_{C_i}))$ is a fan-in of $floor(P_{C_{i+1}})$, therefore

$$l(dup(ceil(P_{C_i}))) + \delta(floor(P_{C_{i+1}})) \leq l_{new}. \tag{9}$$

There are two possible outcomes in Case 2 according to (9). The one is that $l(dup(ceil(P_{C_i}))) + \delta(floor(P_{C_{i+1}})) = l_{new} < l_{\text{old}}$ according to (8). Under this condition, $floor(P_{C_{i+1}})$ is no longer a failing node and therefore P is no longer a failing path. The other is that $l(dup(ceil(P_{C_i}))) + \delta(floor(P_{C_{i+1}})) < l_{\text{new}}$. Under this condition, $dup(ceil(P_{C_i}))$ is no longer a failing node and therefore P is no longer a failing path.

From the analysis of Case 1 and 2, Property 3 is proved.□

Property 4: No new failing path will be introduced after applying Technique 1.

Proof: For $v \in P_{C_i}$, u is a fan-in of v, $u \in C_i$, and $u \notin P_{C_i}$, an extra delay D is added on $\langle u, dup(v)\rangle$. Therefore, any path P' except P changing its way due to the duplication incurs the extra delay D if it passes through u. However, it is found that

$$\begin{aligned} &l(u) + \delta(v) + \Delta(v, ceil(P_{C_i})) + D + \delta(floor(P_{C_{i+1}})) \\ &\quad = l(u) + D + \delta(v) + \Delta(dup(v), dup(ceil(P_{C_i}))) \\ &\qquad + \delta(floor(P_{C_{i+1}})). \end{aligned} \tag{10}$$

Equation (10) indicates the delay between u and $floor(P_{C_{i+1}})$ is not changed after the duplication. In other words, if P' is not a failing path before the duplication, then P' will not be a failing path after the duplication. Therefore, there is no new failing path being introduced after applying Technique 1. □

Fig. 5 illustrates how Technique 1 works. The selected failing path P is shown in bold and the number located inside a node represents the label of that node while the pair of numbers represents the possible range of that label according to (6). After reclustering, P is a failing path only if the label sequence starting from $ceil(P_{C_{i-1}})$ to $floor(P_{C_{i+1}})$ is (4, 7, 8, 9, and 10) along P. However, it will never happen after applying Technique 1. Therefore, it is clear that there exists at least one nonfailing node along P and thus P is no longer a failing path.

After duplicating P_{C_i} to $C_i + 1$, the increase of $W(C_i + 1)$ could be less than $W(P_{C_i})$ because some replicas of nodes $\in P_{C_i}$ may have already existed in $C_i + 1$. In our algorithm, if two copies of an identical node are found within a cluster, the one with the smaller label replaces the other with the larger label. That is, only one copy of a node can exist in a cluster C. As a result, $W(C)$ can be minimized to allow future replications of other nodes into this cluster. It is also noticeable that Technique 1 does not introduce any new cluster.

Technique 2: Duplicate both P_{C_i} and $cone^P_{C_{i+1}}$ to C_d where $1 \leq i \leq k-1$ and $ID(C_i + 1) < ID(C_d) \leq ID(C_i + 2)$ if the capacity constraint of C_d is still met. Then, replace the fan-in of $floor(P_{C_{i+2}})$, that is $ceil(P_{C_{i+1}})$, with $dup(ceil(P_{C_{i+1}}))$ and replace the fan-in of $dup(floor(P_{C_{i+1}}))$, that is $ceil(P_{C_i})$, with $dup(ceil(P_{C_i}))$ as shown in Fig. 6.

Property 5: The target failing path P can be eliminated and no new failing path will be introduced after applying Technique 2.

Proof: Property 5 can be proved in two steps.

Step 1: Duplicate $cone^P_{C_{i+1}}$ to C_d and replace the fan-in of $floor(P_{C_{i+2}})$, that is $ceil(P_{C_{i+1}})$, with $dup(ceil(P_{C_{i+1}}))$. At this point, the failing path P changes its way by passing through C_d instead of C_{i+1}. The circuit delay does not increase because $l(dup(ceil(P_{C_{i+1}})))$ is identical to $l(ceil(P_{C_{i+1}}))$ according to Property 1. For any path P' except P changing its way due to the duplication, if P' is not a failing path before the duplication, then P' will not be a failing path after the duplication according to (3). That is, no new failing path will be introduced.

Step 2: It is clear that duplicating P_{C_i} to C_d and replace the fan-in of $dup(floor(P_{C_{i+1}}))$ with $dup(ceil(P_{C_i}))$ is equivalent to applying Technique 1. Therefore, Property 3 and 4 derived from Technique 1 are also held for Technique 2. □

Fig. 6 illustrates how Technique 2 works. After reclustering, P is a failing path only if the label sequence starting from $ceil(P_{C_{i-1}})$ to $floor(P_{C_{i+2}})$ is (4, 7, 8, 9, 10, and 13) along P. However, it will never happen after applying Technique 2. Therefore, it is clear that there exists at least one nonfailing node along P and thus P is no longer a failing path. Also note that the condition, $ID(C_{i+1}) < ID(C_d) \leq ID(C_{i+2})$, presented in Technique 2 is set to maintain the acyclic property of the current clustering solution S.

Technique 3: Duplicate P_{C_i} and $cone^P_{C_{i+1}}$ to a newly allocated cluster C_d which is inserted into S at some place between C_{i+1} and C_{i+2} if $W(P_{C_i} \cup cone^P_{C_{i+1}})$ does not exceed the capacity constraint. Then, replace the fan-in of $floor(P_{C_{i+2}})$, that is $ceil(P_{C_{i+1}})$, with $dup(ceil(P_{C_{i+1}}))$ and replace the fan-in of $dup(floor(P_{C_{i+1}}))$, that is $ceil(P_{C_i})$, with $dup(ceil(P_{C_i}))$.

Property 6: The target failing path P can be eliminated and no new failing path will be introduced after applying

```
Reclustering(failing_path P) {
    $\{C_1, C_2, \ldots, C_k\} \leftarrow$ a set of clusters which $P$ crosses in topological order;
    for($i = 1$; $i < k-1$; $i$++)
        if($W(P_{C_i} \cup C_{i+1}) \le M$)
            Apply Technique 1; and then return Success;
    for($i = 1$; $i < k-1$; $i$++)
        foreach($C_d$ where $ID(C_{i+1}) < ID(C_d) \le ID(C_{i+2})$)
            if($W(P_{C_i} \cup cone^P_{C_{i+1}} \cup C_d) \le M$)
                Apply Technique 2; and then return Success;
    for($i = 1$; $i < k-1$; $i$++)
        if($W(P_{C_i} \cup cone^P_{C_{i+1}}) \le M$) {
            $C_d \leftarrow$ Allocate a new cluster and insert it immediately before $C_{i+2}$;
            Apply Technique 3; and then return Success;
        }
    return Failure;
}
```

Fig. 7. The area/delay tradeoff reclustering algorithm.

Technique 3.

Proof: Technique 3 is almost identical to Technique 2 except that they encounter different capacity constraints. Technique 2 is applicable only if $W(P_{C_i} \cup cone^P_{C_{i+1}} \cup C_d) \le M$ while Technique 3 is applicable only if $W(P_{C_i} \cup cone^P_{C_{i+1}}) \le M$ because C_d is an empty cluster initially. Thus, the proof is similar to that of Property 5.

The range constraint of C_d is also set to maintain the acyclic property of S. However, an extra cluster is added to S to eliminate the target failing path.

By reviewing three proposed reclustering techniques, it is obvious that the area overhead incurred by applying them increases from Technique 1 to Technique 3. Therefore, to improve the circuit delay without paying too much area overhead, these three techniques are sequentially applied during area/delay tradeoff operations as shown in Fig. 7.

After the reclustering phase, the clustering solution still remains acyclic. If this property must be held for some applications, the algorithm should stop here. Otherwise, a postprocessing step which reduces the number of required clusters without sacrificing the circuit delay is applied. Two techniques, *node removal* and *cluster merging*, are included in our postprocessing step. Both of them were first proposed in [3]. *Node removal* replaces a node v by one of its replicas r if the circuit delay does not increase. *Cluster merging* is a heuristic technique which iteratively merges two clusters to become a new single cluster if the capacity constraint is still met. At each iteration, a pair of mergeable clusters with the largest gain is selected to be merged. The gain is calculated to reflect the benefits obtaining from the corresponding merge. This process is repeated until there is no mergeable pair.

V. Experimental Results

Our area/delay tradeoff reclustering algorithm, denoted as *ADTOC*, has been implemented in SIS environment [8]. In order to evaluate its quality, a set of comprehensive clustering solutions from area-optimized one to the delay-optimized one is produced for each benchmark circuit described in Section III and the results are shown in Table II. Column $L|_A opt$ shows the delay of the initial area-optimized clustering solution. The remaining columns represent the number of clusters required to achieve the designated circuit delay. From Table II, it is clear that our algorithm can actually provide a wide range of clustering solutions which can be selected by designers to perfectly match their specifications.

In order to show how good the delay-optimized results *ADTOC* produces, the results produced by *ADTOC* are shown in Table III. From Table III, it is found that the circuit delay can be effectively reduced with only 9% nodes or 19% clusters increase. Table III also includes the results produced by *CLUSTER-RW* [4] for comparison. From the perspective of delay, *ADTOC* can get the delay optimal solutions for six out of nine benchmark circuits. The average circuit delay obtained by *ADTOC* is only 1.7% larger than that provided by *CLUSTER-RW*. However, from the perspective of area, the node replication ratio is 310% for *CLUSTER-RW* while only 9% for *ADTOC*. On average the numbers of clusters required by *CLUSTER-RW* and *ADTOC* are 300 and 19% more than that of *INITIAL*, respectively. Finally, every circuit takes no more than 1 min to complete the experiment except that c6288 takes 370 s.

Experimental results clearly show that *ADTOC* can effectively produce a complete set of area/delay tradeoff clustering solutions for a given circuit. Moreover, two extreme clustering solutions provided by *ADTOC* are also promising. The delay-considered area-optimized solutions are favorable for designs targeting for small area. The delay-optimized solutions have almost the same performance with much less area overhead as compared with those of the delay optimal solutions reported in [4].

VI. Circuit Clustering under Capacity and Pin Constraints

ADTOC can be easily extended to deal with the clustering problem under both capacity and pin constraints. Almost the

TABLE II
EXPERIMENTAL RESULTS OF OUR AREA/DELAY TRADEOFF CLUSTERING ALGORITHM

		L	L-1	L-2	L-3	L-4	L-5	L-6	L-7	L-8	L-9	L-10	L-11	L-12
circuit	$L\|_{Aopt}$	#clu	#clu	#clu	#clu	#clu	#clu	#clu	#clu	#clu	#clu	#clu	#clu	#clu
c499	15	3	3	3	4	3	-							
c880	25	4	5	5	-									
c1355	29	6	6	6	7	7	-							
c1908	47	9	10	10	10	10	11	11	-					
c2670	42	12	12	12	13	13	13	13	14	14	-			
c3540	61	17	17	17	18	19	19	19	20	20	23	22	23	22
c5315	58	23	23	23	24	24	24	25	25	-				
c7552	50	35	35	35	36	37	38	-						
		L	L-3	L-6	L-9	L-12	L-15	L-18	L-21	L-24	L-27	L-30	L-33	L-34
circuit	L|Aopt	#clu	#clu	#clu	#clu	#clu	#clu	#clu	#clu	#clu	#clu	#clu	#clu	#clu
c6288	166	25	25	25	25	26	28	28	30	32	34	32	34	-

TABLE III
COMPARISONS AMONG *INITIAL, ADTOC,* AND *CLUSTER-RW*

	INITIAL			*ADTOC*			*CLUSTER-RW*		
circuit	delay	#node	#cluster	delay	#node	#cluster	delay	#node	#cluster
c499	15	202	3	11	217	3	11	3200	32
c880	25	357	4	23	415	5	23	848	9
c1355	29	514	6	25	569	7	25	3560	36
c1908	47	880	9	41	950	11	41	4272	44
c2670	42	1161	12	34	1227	14	34	2073	21
c3540	61	1667	17	49	1880	22	48	7496	76
c5315	58	2290	23	51	2311	25	51	8230	83
c6288	166	2416	25	133	3010	34	128	13647	137
c7552	50	3466	35	45	3540	38	44	9786	98
total	493	12953	134	412	14119	159	405	53112	536

entire framework can be directly adopted, except:

1) the initial acyclic clustering solution should satisfy both capacity and pin constraints;
2) the additional pin constraint for the related clusters (C_i and C_{i+1} in Technique 1; C_i, C_{i+1}, C_d, and C_{i+2} in Techniques 2 and 3) must be satisfied while applying three reclustering techniques;
3) both capacity and pin constraints must be met within every postprocessing operation.

After these modifications, *ADTOC* is capable of performing area/delay tradeoff operations under both constraints. Furthermore, the initial clustering solution is obtained by putting each node into a separate cluster after applying the delay-oriented topological sorting. These fine-grained clusters potentially have more flexibility for later reclustering operations, and then lead to better delay-optimized solutions. This extended algorithm is referred to as *PADTOC.* To show its effectiveness, we use it to generate the delay-optimized solutions under the previous parameters plus the extra pin constraint $P = 80$, and the results are given in Table IV. Column *CLUSTER* shows the results provided by an existing algorithm proposed in [7], and Column *PADTOC* shows the results provided by *PADTOC.* The results show that the average circuit delay obtained by *PADTOC* is 1.2% larger than that provided by *CLUSTER.* However, the average number of required clusters is 13% less than that of *CLUSTER.* Besides, the cluster utilization ratio of *PADTOC* is lower than that of *ADTOC.* It means that the pin constraint is the major limit while applying these reclustering techniques. This condition is getting worse as the pin constraint becomes tighter.

VII. CONCLUSIONS

In this paper, we propose an iterative area/delay tradeoff clustering algorithm. The approach begins with finding an initial delay-considered area-optimized acyclic clustering solution. Our reclustering algorithm *ADTOC* is then applied to get the set of comprehensive area/delay tradeoff clustering solutions. Experimental results show that the delay-optimized solutions provided by *ADTOC* have almost the same performance with that of the delay optimal solutions provided by the existing delay optimal algorithm [4] while the required area overhead is significantly reduced. Hence, this algorithm is very

TABLE IV
CLUSTER VERSUS *PADTOC* UNDER
$\delta(v) = 1, w(v) = 1, D = 2, M = 100, P = 80$

	CLUSTER			*PADTOC*		
circuit	delay	#node	#cluster	delay	#node	#cluster
c499	11	202	3	11	259	3
c880	23	409	5	23	413	6
c1355	25	598	7	25	577	7
c1908	41	2390	24	41	1135	15
c2670	34	1545	17	34	1418	21
c3540	48	4924	51	49	2338	36
c5315	51	5713	59	51	2461	58
c6288	128	6233	65	132	3847	43
c7552	44	7746	79	44	4490	80
total	405	29760	310	410	16938	269

useful on solving the timing-driven circuit clustering problem. When both capacity and pin constraints are concerned, *ADTOC* is still capable of providing comprehensive area/delay tradeoff clustering solutions. It also produces promising delay-optimized solutions and keeps lower area overhead while compared with *CLUSTER* [7]. Hence, *ADTOC* can be easily applied to perform area/delay tradeoff clustering operations for multi-FPGA systems.

REFERENCES

[1] F. M. Johannes, "Partitioning of VLSI circuits and systems," in *Proc. 33rd Design Automation Conf.,* June 1996, pp. 83–87.
[2] E. L. Lawler, K. N. Levitt, and J. Turner, "Module clustering to minimize delay in digital networks," *IEEE Trans. Computers,* vol. C-18, no. 1, pp. 47–57, Jan. 1969.
[3] R. Murgai, R. K. Brayton, and A. Sangiovanni-Vincentelli, "On clustering for minimum delay/area," in *Proc. Int. Conf. Computer-Aided Design,* Nov. 1991, pp. 6–9.
[4] R. Rajaraman and D. F. Wong, "Optimal clustering for delay minimization," in *Proc. 30th Design Automation Conf.,* June 1993, pp. 309–314.
[5] R. Aggarwal, R. Murgai, and M. Fujita, "Speeding up technology-independent timing optimization by network partitioning," in *Proc. Int. Conf. Computer-Aided Design,* Nov. 1997, pp. 83–90.
[6] J. Cong, Z. Li, and R. Bagrodia, "Acyclic multi-way partitioning of Boolean networks," in *Proc. 31st Design Automation Conf.,* June 1994, pp. 670–675.
[7] H. Yang and D. F. Wong, "Circuit clustering for delay minimization under area and pin constraints," in *Proc. European Design and Test Conf.,* Mar. 1995, pp. 65–70.
[8] E. M. Sentovich *et al.,* "SIS: A system for sequential circuit synthesis," Electron. Res. Lab. Memo. UCB/ERL M92/41, May 1992.

Juinn-Dar Huang received the B.S. degree in electronics engineering from National Chiao Tung University, Hsinchu, Taiwan, in 1992. He is currently working toward the Ph.D. degree in the Department of Electronics Engineering at the same university.

His current research interests include logic synthesis and circuit partitioning.

Jing-Yang Jou received the B.S. degree in electrical engineering from National Taiwan University, Taiwan, and the M.S. and Ph.D. degrees in computer science from the University of Illinois at Urbana-Champaign.

He is a Professor in the Department of Electronics Engineering at National Chiao Tung University, Hsinchu, Taiwan. He has worked in the GTE Laboratories and Bell Laboratories. His research interests include behavioral and logic synthesis, VLSI designs and CAD for low power, design verification, synthesis and design for testability, and hardware/software codesign. He has published more than 50 journal and conference papers.

Dr. Jou is a member of Tau Beta Pi, and the recipient of the Distinguished Paper Award of the IEEE International Conference on Computer-Aided Design in 1990. He served as the Technical Programmer Chair of the Asia-Pacific Conference on Hardware Description Languages (APCHDL'97) and

Wen-Zen Shen received the Ph.D. degree in electronics engineering from National Chiao Tung University, Hsinchu, Taiwan, in 1982.

He is currently a Professor in the Department of Electronics Engineering, National Chiao Tung University. His current research interests include VLSI design, low-power circuit design, computer-aided design, and VLSI for signal processing.

Dr. Shen is a member of Phi Tau Phi.

Hsien-Ho Chuang received the B.S. degree in electrical engineering from National Taiwan University, Taiwan, in 1991. He is currently working towards the Ph.D. degree in the Department of Electronics Engineering, National Chiao Tung University, Hsinchu, Taiwan.

His current research interests include the various challenges on the look-up table-based FPGA, such as logic synthesis, architecture design, hardware emulation, and partitioning.