Performance-Driven Multi-Level Clustering for Combinational Circuits

C. N. Sze  
Department of Electrical Engineering  
Texas A&M University  
College Station, Texas 77843-3259, USA  
e-mail: cnsze@ee.tamu.edu

Ting-Chi Wang  
Department of Computer Science  
National Tsing Hua University  
Hsinchu, Taiwan  
e-mail: tcwang@cs.nthu.edu.tw

Abstract—In this paper, an effective algorithm is presented for performance driven multi-level clustering for combinational circuits, and is applicable to hierarchical FPGAs. With a novel graph contraction technique, which allows some crucial delay information of a lower-level clustering to be maintained in the contracted graph, our algorithm recursively divides the lower-level clustering into the next higher-level one in a way that each recursive clustering step is accomplished by applying a modified single-level circuit clustering algorithm based on [1]. We test our algorithm on the two-level clustering problem and compare it with the latest algorithm in [2]. Experimental results show that our algorithm achieves, on average, 12% more delay reduction when compared to the best results (from TLC with full node-duplication) in [2]. In fact, our algorithm is the first one for the general multi-level circuit clustering problem with more than two levels.

I. INTRODUCTION

Circuit clustering is defined as assigning circuit elements into clusters under different design constraints, such as area and pin constraints [1, 3, 4, 5]. In this way, the circuit clusters are smaller compared to the original circuit, and hence manipulation and synthesis of the clusters are easier. Most circuit clustering algorithms aim at either minimizing the circuit delay or the inter-cluster connections.

In this paper, we focus on the problem of combinational circuit clustering for delay minimization subject to area constraints. This problem is first studied in [5]. The authors formulate the problem in the unit delay model in which no delay value is associated with any connection within a cluster or with any gate while unit delay is assigned to each inter-cluster connection. A polynomial time algorithm is also proposed to solve the problem optimally. Recently, most research adopts the general delay model [4], in which each gate is associated with a delay value, no delay is for each connection within the same cluster, and a constant delay is for each inter-cluster connection. An algorithm which solves the circuit clustering problem based on the general delay model is proposed in [1]. It is proved that the algorithm can optimally solve the problem in polynomial time. The problems considered in [1, 4, 5] are referred to as single-level circuit clustering.

The necessity of a solution to the multi-level circuit clustering problem is increasing when more and more designs are built on hierarchical FPGA architectures. The two-level clustering problem with area constraints is studied in [2]. The problem formulation requires the division of a circuit into clusters (second-level clusters) and each cluster is then further divided into smaller clusters (first-level clusters). It is proved that two-level circuit clustering for delay minimization is NP-hard. Hence, they propose a heuristic which is extended from [1]. Their algorithm constructs a candidate second-level cluster rooted at each node and then covers the whole circuit based on the clusters. During the construction of each candidate second-level cluster, the first-level clusters within it are formed at the same time. Both first-level and second-level clusters are constructed according to the same criterion—nodes are chosen by comparing the maximum delay of the paths from primary inputs to the cluster root passing through them.

However, the heuristic in [2] is not effective enough according to our experiments. The main reason may be related to the restriction in which it does not allow node duplication within a second-level cluster. Node duplication within a second-level cluster has two contrasting effects on delay minimization. On one hand, it may reduce the circuit delay since a node can be included into different clusters so that the number of inter-cluster connections may be reduced. On the other hand, each cluster is constrained by an area bound and node duplication consumes area, so less different nodes can be included into a second-level cluster and then the circuit delay may increase. However, we have shown by experiment that properly allowing node duplication within second-level clusters is beneficial to delay minimization. Moreover, since the algorithm in [2] performs first-level and second-level clusterings at the same time, for each node included into a first-level cluster, all the data for first-level and second-level clusters (e.g., lists for candidate nodes and immediate successor with maximum delay) should be updated accordingly. It makes their algorithm hardly extensible to solve the circuit clustering problem with more than two levels.

In order to cope with the difficulties mentioned above, we propose an algorithm for the general combinational circuit clustering problem with any arbitrary number of levels. Our algorithm constructs clusters for each level separately, from the first level to the desired level. The clustering of each level is performed on a contracted graph which only captures the most important delay information from the clustering of the previous level. Besides, since we only perform circuit clustering on the contracted graph formed from the previous level, a simple but effective single-level circuit clustering algorithm can be employed. As a result, our algorithm effectively handles the multi-level problem by repeating the graph contraction technique and the single-level circuit clustering algorithm.

Although we employ a single-level clustering algorithm which is extended from [1], our overall algorithm is not merely a trivial extension of [1], because without our graph contraction technique, the single-level clustering algorithm cannot be repeatedly applied to the circuit to obtain a multi-level clustering. As a result, the graph contraction algorithm plays a critical role in our work and it successfully links every two successive levels of
circuit clustering.

Taking the two-level clustering problem as an example, our algorithm first divides the circuit into a set of first-level clusters with node duplication, so that the node duplication within second-level clusters can be later guided by those first-level clusters. In this way, node duplication within a second-level cluster happens only when the duplication helps minimizing the delay of the resultant first-level clustering. In fact, our implementation and experimental results show that, in this mechanism, allowing node duplication within second-level clusters indeed further reduces the delay values. We are able to achieve 12% more delay reduction when comparing with the algorithm in [2], in which node duplication within the second-level clusters is not allowed.

II. DEFINITIONS AND PROBLEM FORMULATION

A combinational circuit can be represented as a directed acyclic graph (DAG) \( G = (V, E) \). \( V \) is the set of nodes which represent the functional blocks (e.g., gates) in the circuit and \( E \) is the set of edges which stand for the connections among the blocks. In the graph, PIs have in-coming edges only, and on the contrary, P0s have in-coming edges only.

An area function \( w(v) \) is defined for each node \( v \in V \). The value of \( w(v) \) represents the area of the corresponding functional block.

A first-level cluster \( C^1 \subseteq V \) is a set of nodes \( \{v_1, v_2, ..., v_k\} \) which satisfies the first-level area bound \( M_1 \), and a second-level cluster \( C^2 \) is a set of first-level clusters \( \{C^1_1, C^1_2, ..., C^1_k\} \) which satisfies the second-level area bound \( M_2 \). General, an \( i \)-th level cluster \( C^i \) is a set of \((i-1)\)-th level clusters \( \{C^{i-1}_1, C^{i-1}_2, ..., C^{i-1}_k\} \) and its area bound is denoted as \( M_i \). For each first-level (second-level) cluster, its area function is defined as the sum of area of all nodes (the sum of first-level area bounds) in the cluster. That is,

\[
w(C^1) = \sum_{v \in C^1} w(v) \quad \text{and} \quad w(C^2) = \sum_{C^1 \in C^2} M_1
\]

In general, for each \( i \)-th level cluster \( C^i \), we have

\[
w(C^i) = \sum_{C^{i-1} \in C^i} M_{i-1}, \quad i \in \{2, ..., n\}
\]

where \( n \) is the desired level of circuit clustering.

In the definition of the cluster area for an \( i \)-th level cluster \( C^i \), all \((i-1)\)-th level clusters inside \( C^i \) are considered to have the same area \( M_{i-1} \). So totally no more than \( M_n \) \((i-1)\)-th level clusters can be included into one \( i \)-th level cluster\(^1\).

Besides area constraints, there are delay values associated with all nodes and edges. For each node \( v \in V \), a delay function \( \delta(v) \) is defined as a non-negative value which represents the delay of the functional block. The notation \( \delta(a, b) \) represents the edge delay from node \( a \) to node \( b \). For each edge within the same first-level cluster, it is associated with a fixed delay \( D_1 \). For each edge connecting two nodes in different first-level clusters, but in the same second-level cluster, the edge delay is assigned to a fixed delay \( D_2 \). Generally, for each edge connecting two nodes in different \((i-1)\)-th level clusters, but in the same \( i \)-th level cluster, the edge delay is associated with a fixed delay \( D_i \). And, each edge, which connects two nodes between two different \( n \)-th level clusters, has a fixed delay \( D_{n+1} \). Practically, we have \( D_1 < D_2 < ... < D_{n+1} \) for an \( n \)-level circuit clustering.

For the delay of a path from node \( a \) to node \( b \) we always include all node delays and edge delays along the path. The path delay at a node \( v \) is defined as the maximum delay of all paths from PIs to \( v \). The delay of a clustered circuit is defined as the maximum path delay at all PO nodes; in other words, it is the maximum delay of all paths from PIs to POs within the clustered circuit. According to the above definitions, the multi-level circuit clustering problem with area constraints is presented in the following.

Problem (Multi-Level Circuit Clustering)

Divide the graph \( G \) into a set \( S_1 = \{C^1_1, C^1_2, ..., C^1_k\} \) of first-level clusters, divide the set of all first-level clusters into a set \( S_2 = \{C^2_1, C^2_2, ..., C^2_m\} \) of second-level clusters, and recursively divide all the \((i-1)\)-th level clusters into a set \( S_i = \{C^i_1, C^i_2, ..., C^i_n\} \) of \( i \)-th level clusters until a set of n-th level clusters is obtained, such that the delay of the clustered circuit is minimized. The clusters of each level may have common elements, but the clustered circuit must be logically equivalent to the original circuit. The clusters of each level should satisfy the following conditions.

\[
\forall j \in \{1, ..., m_1\}, C^1_j \subseteq V, \text{ s.t. } \begin{cases} w(C^1_j) \leq M_1, \\ \bigcup_{j=1}^{m_1} C^1_j = V \end{cases}
\]

\[
\forall j \in \{1, ..., m_2\}, C^2_j \subseteq S_1, \text{ s.t. } \begin{cases} w(C^2_j) \leq M_2, \\ \bigcup_{j=1}^{m_2} C^2_j = S_1 \end{cases}
\]

and,

\[
\forall i \in \{3, ..., n\}, \forall j \in \{1, ..., m_i\}, C^i_j \subseteq S_{i-1}, \text{ s.t. } \begin{cases} w(C^i_j) \leq M_i, \\ \bigcup_{j=1}^{m_i} C^i_j = S_{i-1} \end{cases}
\]

\(^1\)It seems that the area definitions of clusters are different from [2], but in fact, the TL-C implementations we obtained from the authors of [2] follow our definitions here. Details are discussed in Section VI.

---

![Fig. 1. An example of circuit clustering with three levels](image-url)
An example is shown in Figure 1. In the figure, there are 17 nodes in the graph and a three-level clustering is shown. In the clustering, the delay of edge \((j, k)\) is \(D_4\) since \(j\) and \(k\) are in different third-level clusters. The delay values of \((l, q)\) and \((i, j)\) are \(D_3\) and \(D_2\) respectively. Since \(e\) and \(f\) are in the same first-level cluster, the delay associated with \((e, f)\) is \(D_1\). If each node is associated with unit area and \(M_1 = 2\), \(M_2 = 6\), \(M_3 = 12\), the area \(w\) of all first-level clusters in the clustering is 2 except for the clusters containing \(i, j\) or \(o\), whose \(w = 1\). The second-level cluster containing \(\{a, b, c, d, e, f\}\) has an area of 6. The area of the second-level cluster containing \(\{g, h, i\}\) is also 6 since it consists of three first-level clusters and no more first-level cluster can be further filled into it. In fact, the area of all second-level clusters in Figure 1 is 6 except the one containing \(\{k, l\}\) (whose area is 2). The area of each third-level cluster in the graph is the same, which is 12.

In the example, we assume \(D_1 = 1\), \(D_2 = 3\), \(D_3 = 7\), \(D_4 = 17\) and \(\delta(v) = 1\) for each node \(v\). The delay of the clustered circuit is equal to 85, which is along the path \(a \rightarrow b \rightarrow e \rightarrow f \rightarrow g \rightarrow h \rightarrow j \rightarrow k \rightarrow l \rightarrow q\), including the edge entering \(a\) and the one leaving \(q\) (The delays of both are \(D_4\)).

III. THE ALGORITHM

The flow of our algorithm is depicted in Figure 2. The algorithm mainly consists of two parts: single-level graph clustering and graph contraction.

![Flow of the algorithm](image)

At the beginning of our algorithm, the level index (denoted by \(le\)) is set to 0, which means the circuit is not yet clustered. Then, single-level graph clustering is performed on the input graph \(G_{lev} = G\) when \(lev = 0\) and it outputs a set of clusters, \(S_{lev+1}\) (= \(S_1\) for the first time). After that, \(lev\) is increased by 1. If \(lev\) is equal to the number of levels, \(n\), the algorithm terminates. Otherwise, graph contraction is performed on \(G_{lev+1}\) (= \(G_0\) for the first time) based on \(S_{lev}\). To generate the contracted graph \(G_{lev}\), we treat each \(lev\)-th-level cluster \(C_{lev}\) (at the first iteration, we have the set \(S_1\) of first-level clusters) as an independent supernode. Thus, \(G_{lev}\) is built in the way that each node represents a supernode, which stands for a cluster in \(S_{lev}\). Then single-level graph clustering is performed on the new graph \(G_{lev}\), and the output is a set \(S_{lev+1}\) of clusters. The algorithm is iterated until the desire level \(n\) of clusters is obtained.

The details of the single-level graph clustering and graph contraction algorithms are presented in Section 3.1 and 3.2 while the graph contraction algorithm is the main contribution in the paper since without graph contraction, single-level graph clustering cannot be repeatedly applied to the circuit \(G\).

A. Single-Level Graph Clustering

In the algorithm, given a graph \(G_{lev}\), clustering \(S_{lev+1}\) is constructed by applying the single-level circuit clustering algorithm in [1] with modifications. The delay model adopted in [1] is slightly different from the delay model that we consider in this paper. There is no delay value (e.g., \(D_1 = 0\)) for each edge within the same cluster in the delay model in [1]. But, in our model, the delay of each edge within the same cluster can be any non-negative number. (Note that \(\delta_1(e) = D_1\) for each edge \(e\) at the beginning of the algorithm when \(lev = 0\). After the first iteration, \(lev \geq 1\), each edge delay is assigned by graph contraction and it may not be a fixed value for every edge. Here, we only use \(D_1\) for the illustration of the delay model difference.) However, if we assume each edge has the delay \(D_1\) during the calculation of the delay matrix \(\Delta_1\), which stores the maximum delay (including node delays and edge delays) of the paths between any two nodes, single-level graph clustering under our model can be solved optimally in a similar way. The pseudo-code of the modified single-level graph clustering algorithm is shown in the following.

```
ALGORITHM Single-Level Circuit Clustering(G_{lev})
Input : graph G_{lev} = \{V_{lev}, E_{lev}\}
Output : clustering S_{lev+1} = \{C_{lev+1}^1, C_{lev+1}^2, ..., C_{lev+1}^{k_{lev+1}}\}
1. begin
2. compute the delay matrix \(\Delta_{lev+1}\), where \(\Delta_{lev+1}(i, j)\) is the maximum delay (including node and edge delays) of the paths from \(i\) to \(j\);
3. FOR each PI \(i\), DO \((i, \delta_{lev+1}(i))\);
4. Sort the non-PI nodes of \(G_{lev}\) in a topological order to obtain list \(T\);
5. WHILE \(T\) is not empty
6. \(FIR\) the first node \(v\) from \(T\);
7. Compute \(N_v\);
8. FOR each node \(u\) \(\in\) \(N_v\) \(-\{v\}\) do
9. \(f(u) = f(u) + \Delta_{lev+1}(u, v)\);
10. end FOR
11. end WHILE
12. \(L =\) all PD nodes;
13. \(S_{lev+1} = \varnothing;\)
14. WHILE \(L\) is not empty
15. \(FIR\) a node \(v\) from \(L\);
16. \(S_{lev+1} = S_{lev+1} \cup \{\text{cluster}(v)\};\)
17. \(FIR\) all nodes \(z\) \(\in\) \(V_{lev}\) \(-\{v\}\) that \(z\) is an input of \(\text{cluster}(v)\) and \(\text{cluster}(z)\) \(\notin S_{lev+1}\);
18. \(L = L \cup \{z\};\)
19. end WHILE
20. end;
21. begin
22. \(\text{cluster}(v) = \{v\};\)
23. WHILE \(P\) is not empty
24. \(\text{FIR} the first node \(u\) in \(P\);
25. \(f(u) = f(u) + \Delta_{lev+1}(u, v)\);
26. \(\text{cluster}(v) = \text{cluster}(v) \cup \{u\};\)
27. ELSE
28. interface;
29. end IF
30. end WHILE
31. end;
32. end.
```

In this algorithm, \(N_v\) is defined as the set of all predecessors of \(v\) together with \(v\) in the graph. Each cluster has exactly one root node and we denote the cluster rooted at \(v\) as \(\text{cluster}(v)\). All nodes in \(\text{cluster}(v)\) are also in \(N_v\).

At the first time the algorithm is performed, the circuit is un-clustered. We have \(lev = 0\), \(\delta_1(v) = \delta(v)\) for each node \(v\), and \(\delta_1(x, y) = \delta(x, y) = D_1\) for each edge \((x, y)\) in the graph \(G_0\). After the first time (\(lev \geq 1\)), the values of \(\delta_{lev+1}(v)\) and \(\delta_{lev+1}(x, y)\)
are determined in the graph contraction step (discussed in the next section). Moreover, the calculation of delay matrix $\Delta_{\text{lev+1}}$ is based on the $\delta_{\text{lev+1}}(x, y)$ values in the input graph $G_{\text{lev}}$.

In Single-Level Circuit Clustering, lines 1-11 describe the labeling phase: for each node $v$, a best cluster rooted at $v$ is calculated, and the corresponding path delay at $v$ (namely the label $l_{\text{lev+1}}(v)$ of $v$) is recorded. The cluster and label calculation for each node is in the function "Labeling". Lines 12-19 in Single-Level Circuit Clustering is the clustering phase. The clusters are built from POS until the whole circuit is covered by them.

### B. Graph Contraction

After constructing the set $S_{\text{lev}}$ of lev-th level clusters, we build a contracted graph $G_{\text{lev}}$ from the graph $G_{\text{lev+1}}$ and the clustering $S_{\text{lev}}$. The contracted graph $G_{\text{lev}}$ is constructed such that each node $v_i$ in $G_{\text{lev}}$ corresponds to a lev-th level cluster $C_i^{\text{lev}}$ in $S_{\text{lev}}$ and its path delay in $G_{\text{lev}}$ is the same as that of the root node of $C_i^{\text{lev}}$. Since different clusters have different numbers of elements and subgraph structures, and there are different edges connecting the nodes of two clusters, delay assignment to each supernode and each edge connecting the supernodes is not straightforward.

The general algorithm for constructing $G_{\text{lev}}$ from graph $G_{\text{lev+1}}$ and clustering $S_{\text{lev}}$ is shown in the following.

**Algorithm: Graph Contraction ($G_{\text{lev+1}}$, $S_{\text{lev}}$)**

**Input:** Graph $G_{\text{lev+1}}$, Clustering $S_{\text{lev}} = \{C_i^{\text{lev}}, C_2^{\text{lev}}, ..., C_n^{\text{lev}}\}$

**Output:** $G_{\text{lev}}$

1. FOR each cluster $C_i^{\text{lev}}$ in $S_{\text{lev}}$
2. construct a node $v_i$ in $G_{\text{lev}}$
3. $\delta_{\text{lev+1}}(v_i) = \text{Delay}(C_i^{\text{lev}})$
   /\  $S_{\text{lev+1}}(v_i)$ is assigned in $G_{\text{lev}}$ \/
4. END FOR
5. FOR each edge $e = (a, b)$ in $G_{\text{lev+1}}$
6. IF ($a$ and $b$ are not in the same cluster of $S_{\text{lev}}$)
7. assume $a \in C_i^{\text{lev}}$ which is rooted at $v_i$
8. find the cluster $C_j^{\text{lev}}$ rooted at $v_j$
9. IF (there exists an edge from $v_j$ to $v_i$ in $G_{\text{lev+1}}$)
10. add an edge from $v_j$ to $v_i$ in $G_{\text{lev}}$
11. $\delta_{\text{lev+1}}(v_i, v_j) = \delta_{\text{lev+1}}(a, b) + (D_v - D_u)$
12. /\  Note that $\delta_{\text{lev+1}}(v_i, v_j)$ refers to $G_{\text{lev}}$ \/
13. /\  and $\delta_{\text{lev+1}}(v_i, v_j)$ refers to $G_{\text{lev+1}}$ \/
14. ELSE IF (there exists an edge from $v_i$ to $v_j$ in $G_{\text{lev+1}}$)
15. IF ($\delta_{\text{lev+1}}(v_j, v_i) < \delta_{\text{lev+1}}(a, b) + (D_v - D_u)$)
16. $\delta_{\text{lev+1}}(v_i, v_j) = \delta_{\text{lev+1}}(a, b) + (D_v - D_u)$
17. END IF
18. END IF
19. END FOR
20. END FOR
21. return $G_{\text{lev}}$

In the graph contraction algorithm, in the first FOR loop (lines 1-4), each new node (or namely supernodes) $v_i$ in $G_{\text{lev}}$ is first created, which is corresponding to each cluster $C_i^{\text{lev}}$ in $S_{\text{lev}}$. The delay of each new node $v_i$ is set to $D(C_i^{\text{lev}})$, which is defined as the maximum delay of the paths from any node within $G_i^{\text{lev}}$ to the root of the cluster, in order to keep the maximum delay values of the clusters in the new graph.

The second FOR loop (lines 5-20) creates new edges between the supernodes and to calculate the delay value for each edge. For each inter-cluster edge, we build an edge between the two corresponding supernodes and assign the delay value such that the maximum delay of the paths passing through that inter-cluster edge is maintained. This is the most complicated part for the graph contraction since different edges may connect different nodes of the two clusters. The delay assignment to each edge (lines 9-18) is depicted in Figure 3. (Note that the notation $\delta_{\text{lev}}(a, b)$ represents the edge delay between nodes $a$ and $b$ in graph $G_{\text{lev+1}}$. At the first time graph contraction is performed, $\delta_{1}(a, b)$ is set to $D_1$ for each edge $(a, b)$ in $G_0$. For the case where $a$ and $b$ are nodes generated by the previous graph contraction step, $\delta_{\text{lev}}(a, b)$ is already assigned during the last graph contraction.)

In Figure 3, the "height" of a node, an edge or a cluster represents its delay. It is obvious that $x + \delta_{\text{lev+1}}(v_{ij}) = \delta_{\text{lev}}(a, b) + \Delta_{\text{lev}}(b, r)$, so we have $x = \delta_{\text{lev+1}}(v_{ij}) - \delta_{\text{lev+1}}(v_{ij}) + \Delta_{\text{lev}}(b, r)$. Since the edge between $a$ and $b$ becomes an inter-cluster edge connecting two lev-th-level clusters (originally it connects two (lev-1)-th level clusters), $\delta_{\text{lev+1}}(v_{ij}, v_{ij}) = x + \Delta_{\text{lev+1}} - D_{\text{lev+1}}$ and then we can easily derive the equation in line 11 of **Graph Contraction**.

Finally, for every two supernodes, if there exists more than one edge between them, we only keep the edge with maximum delay value and remove all the others in the new graph $G_{\text{lev}}$.

Note that in the clustering generated by our modified single-level graph clustering algorithm mentioned in Section A, inter-cluster edges only connect from the roots of the predecessor clusters.

**We describe in Section IV that in this graph contraction algorithm, some crucial delay information of each new node $v_e$ is extracted from the root of the corresponding cluster $C_e^{\text{lev}}$. Since this delay information is retained in the contracted graph, finding the next higher-level clustering from the contracted graph for delay minimization can be achieved by the modified single-level graph clustering algorithm in polynomial time.**

### C. Remarks

A multi-level clustering is achieved when we iteratively perform the single-level graph clustering and graph contraction. At first, we set $\text{lev} = 0$ and $G_0 = G$ which is the target unclustered circuit. After the single-level graph clustering process, the first-level clustering $S_1$ is obtained and we can produce $G_1$ by the graph contraction algorithm. Then, a second-level clustering $S_2$ can be found from the contracted graph $G_1$ by the same graph clustering algorithm with $\text{lev} = 1$. This time, the algorithm takes $D_3 - D_2$ for the calculation of $E_2$ value, (in line 10 of "Labeling"), $M_2$ for area bound (in line 5 of "Labeling"), and edge and node delays in $G_1$ for the calculation of $D_2$. After the second-level clustering $S_2$ (or in general, the $i$-th level clustering $S_i$) is obtained, a contracted graph $G_2 (G_i)$ is constructed. The third-level clustering $S_3$ $(i+1)$-th level clustering $S_{i+1}$ can then be generated similarly. It is obvious that our overall algorithm can be employed recursively to get an $n$-level circuit clustering.
IV. Analysis of the Algorithm

In our algorithm, the contracted graph $G_{lew}$ is constructed such that the path delay at $v_l$ in $G_{lew}$ equals the label value $l_{lew}(t)$ of the root node $r$ of the corresponding cluster $C_i^{lew}$ in $G_{lew-1}$. We have the property stated in the following theorem.

**Theorem 1** For any node $v_l$ in the contracted graph $G_{lew} = \{V_{lew}, E_{lew}\}$ generated by the algorithm **Graph Contraction**, the path delay at $v_l$ in $V_{lew}$ is equal to the label, $l_{lew}(v_l)$, of the root node $r$ of the corresponding cluster $C_i^{lew} \in S_{lew}$.

**Proof** It is omitted due to limited space.

Theorem 1 implies that the single-level graph clustering algorithm can be repeatedly applied to cluster the circuit to any level $n$ with correct delay information. With Theorem 1, we can further derive the **local optimality** of our algorithm as stated in the following theorem.

**Theorem 2** Given an $i$-th-level clustering produced by our algorithm, our algorithm generates a $(i+1)$-th-level clustering which divides the $i$-th level clusters and minimizes the delay of the resultant circuit.

**Proof** It is omitted due to limited space.

The local optimality described in Theorem 2 does not guarantee a final globally optimal solution but the local optimality of each recursive step tends to maintain the circuit with a small delay value. In fact, our experiments have shown that the delay reduction achieved by our algorithm is much better than the state-of-the-art algorithms.

For the time complexity of the algorithm, the modified single-level graph clustering algorithm takes $O(V^3\log(V) + V|E|)$ time [1], and each graph contraction takes $O(|V| + |E|)$ time, where $V$ and $E$ are the node and edge sets of a given graph, respectively. For each graph contraction, the number of edges and number of vertices both are non-increasing. So for each level, the complexity of single-level graph clustering algorithm should be still bounded above by $O(V^3\log(V) + V|E|)$. Therefore, the overall time complexity of our algorithm is $O(nV^3\log(V) + V|E|)$ for the multi-level circuit clustering problem with $n$ levels.

V. Postprocessing Techniques

In the experiment, our algorithm generates large clustered circuits since node duplications occur frequently in order to minimize the circuit delay. So, we present two simple postprocessing techniques to reduce the area of a clustered circuit. The techniques are employed after the clustered circuit is generated, and they do not increase the delay of the clustered circuit. Assuming we have an $n$-level clustered circuit, the first technique locates those $n$-level clusters each of which is a subset of another $n$-level cluster and so they can be deleted without changing the delay and functionality of the whole circuit.

The second technique packs several $n$-level clusters into one single cluster if the area constraint is not violated. This can be done when the original clusters are small. The technique is based on the First Fit Decreasing heuristic for the bin packing problem (which is also mentioned in [4]). All the $n$-level clusters are sorted in non-increasing order. We assume each bin has the capacity $M_b$. Then, it starts to place clusters one by one into the bins. Each time we place a cluster in the leftmost bin that still has enough space for it, and start a new bin if necessary.

VI. Implementation and Experimental Results

We tested our algorithm upon a two-level hierarchy which is based on Altera’s APEX FPGA architecture [2] and compared our algorithm (namely MLC) to the UCLA TLC implementations which are obtained from the authors of [2]. Based on the timing extraction done in [2], we used the same parameters: $M_1 = 10$, $M_2 = 160$, $D_1 = 0.36\,\mu s$, $D_2 = 0.85\,\mu s$, $\delta(v_l) = 0.6\,\mu s$, $\forall v_l \in V_{lew}$. We evaluated our algorithm on this 2-level hierarchy mainly because this FPGA architecture is the latest FPGA model for the multi-level circuit clustering problem.

Experiments were performed on MCNC benchmark circuits which also were used by UCLA TLC [2]. The benchmarks were pre-processed and mapped into 4-input LUT networks by UCB SIS and UCLA RAS systems. Each benchmark circuit was clustered into a two-level clustering by the two TLC implementations (No node duplication and Full node duplication) and our algorithm. The first TLC implementation did not allow any node duplication among different second-level clusters while the latter one did. However, both TLC implementations did not allocate duplication within a second-level cluster. The implementations returned a new circuit (if node duplication happens) which is functionally equivalent to the original circuit and also returned the clustering information.

In order to carry out a fair and objective comparison, our implementation strictly followed the same problem formulation as the TLC implementations in [2]. First, in our implementation, each PI node or PO node formed a second-level cluster by itself and it was excluded from any cluster rooted at any other node which is neither PI node nor PO node. As a result, the edge delay from any PI node to any of its immediate successors was always $D_{b}$, while similarly the same delay was always associated with the edge from any node to a PO node. Secondly, our problem formulation and the problem formulation in [2] seemed to be different in the area calculation of second-level clusters. We limited the total number of first-level clusters within a second-level cluster while [2] limited the total number of nodes in a second-level cluster. So, if some first-level clusters contain fewer nodes, more first-level clusters can be included into a second-level cluster in the problem formulation of [2]. While in our problem formulation, the maximum number of first-level clusters within a second-level cluster was always fixed. However, our problem formulation was more appropriate to the APEX FPGA architecture where one MegALAB (=second-level cluster) can hold no more than 16 logic array blocks, LABs (=first-level clusters), even when some of the LABs are not full. In fact, we found that the TLC implementations we obtained from the authors of [2] followed our definition on the second-level cluster area bound, so this ensured a fair comparison.

In our implementation, we have also imposed a constraint on the maximum number of inputs for each first-level cluster. In the specification of Altera APEX FPGA devices, a first-level cluster (LAB) cannot have more than 22 inputs. Hence, we controlled the number of inputs to a first-level cluster by adding a condition in the IF statement in line 5 of “Labeling”, such that we stopped adding new nodes into each first-level cluster when the number of inputs to the cluster was more than 22. This constraint was also included in the UCLA TLC implementations.

The experimental results are shown in Table I. Columns 2-4 show the results of TLC [2] with no node duplication. Columns 5-8 show the best results of TLC in which node duplication is al-
TABLE I
COMPARISON BETWEEN TWO TLC IMPLEMENTATIONS AND OUR ALGORITHM

<table>
<thead>
<tr>
<th>Circuits</th>
<th>UCLA TLC (No ND)</th>
<th>UCLA TLC (Full ND)</th>
<th>Our Algorithm MLC</th>
<th>1-level</th>
</tr>
</thead>
<tbody>
<tr>
<td>delay</td>
<td>CPU</td>
<td>area</td>
<td>delay</td>
<td>%de</td>
</tr>
<tr>
<td>245-38</td>
<td>23.6</td>
<td>0.26</td>
<td>22</td>
<td>21.22</td>
</tr>
<tr>
<td>apex2</td>
<td>14.90</td>
<td>0.14</td>
<td>141</td>
<td>14.92</td>
</tr>
<tr>
<td>apex6</td>
<td>10.79</td>
<td>0.29</td>
<td>50</td>
<td>10.86</td>
</tr>
<tr>
<td>C1085</td>
<td>15.66</td>
<td>0.22</td>
<td>25</td>
<td>15.83</td>
</tr>
<tr>
<td>C3515</td>
<td>16.35</td>
<td>1.0</td>
<td>20</td>
<td>14.4</td>
</tr>
<tr>
<td>C850</td>
<td>18.52</td>
<td>0.14</td>
<td>20</td>
<td>16.35</td>
</tr>
<tr>
<td>das</td>
<td>12.72</td>
<td>0.25</td>
<td>25</td>
<td>11.28</td>
</tr>
<tr>
<td>das</td>
<td>12.72</td>
<td>0.25</td>
<td>25</td>
<td>11.28</td>
</tr>
<tr>
<td>t10</td>
<td>23.89</td>
<td>1.0</td>
<td>20</td>
<td>22.45</td>
</tr>
<tr>
<td>19</td>
<td>11.03</td>
<td>0.19</td>
<td>63</td>
<td>10.31</td>
</tr>
<tr>
<td>L2</td>
<td>14.67</td>
<td>0.37</td>
<td>48</td>
<td>13.95</td>
</tr>
<tr>
<td>large</td>
<td>16.36</td>
<td>0.69</td>
<td>114</td>
<td>15.43</td>
</tr>
<tr>
<td>misex3</td>
<td>14.18</td>
<td>0.69</td>
<td>45</td>
<td>12.97</td>
</tr>
<tr>
<td>too0large</td>
<td>12.51</td>
<td>0.15</td>
<td>3</td>
<td>12.51</td>
</tr>
<tr>
<td>vda</td>
<td>11.77</td>
<td>0.18</td>
<td>39</td>
<td>10.36</td>
</tr>
<tr>
<td>sl</td>
<td>9.08</td>
<td>0.22</td>
<td>50</td>
<td>8.12</td>
</tr>
<tr>
<td>Average</td>
<td>9.54</td>
<td>1.35</td>
<td>8.1</td>
<td>8.1</td>
</tr>
<tr>
<td>Total</td>
<td>18.44</td>
<td>1.388</td>
<td>11.7</td>
<td>11.7</td>
</tr>
</tbody>
</table>

In fact, we can take these 1-level clustering results as a “loose” lower bound for any two-level clustering. From the last column, it is shown that our results produce only 3.5% more delay than the optimal 1-level results. In fact, out of 16 benchmarks, we obtain optimal 2-level clustering solutions for at least 4 circuits (where “%di” values equal to 0.0).

ACKNOWLEDGMENTS

The authors would like to thank Dr. Jason Cong and Michail Romes, for providing their TLC implementations and pre-processed MCNC benchmarks for comparison, and for offering us many helpful discussions.

REFERENCES