Performance-Driven Recursive Multi-Level Clustering

Mehrdad Eslami Dehkordi and Stephen D. Brown
Department of Electrical and Computer Engineering
University of Toronto, Toronto, Canada M5S 3G4
{eslami,brown}@eecg.toronto.edu

Abstract

This paper presents an effective algorithm for multi-level circuit clustering for delay minimization, which is applicable to FPGAs. At the highest level of clustering, the circuit is clustered using a modified single-level clustering algorithm based on [6]. A cluster to netlist transformation technique is proposed, which converts each cluster into a new subcircuit. The algorithm then continues recursively by clustering the generated subcircuits into further levels of clusters. To reduce the amount of area overhead, a node duplication control algorithm based on the node slack is proposed and a cluster packing algorithm is used to reduce the number of clusters. Experimental results on the two-level clustering problem using Quartus Design System from Altera show that our algorithm reduces the delay, on average, by 7.3% compared with the best results in [2]. Also the total FPGA compile time for all the benchmark circuits reported by Quartus is reduced by 36%.

1. Introduction

Circuit clustering is one of the crucial steps in circuit layout for handling large designs. It is defined as dividing a circuit into clusters while satisfying certain design constraints. In performance driven clustering, the problem is to cluster a circuit so as to minimize its delay, subject to some constraints (area and/or pin) on the clusters. A polynomial time optimal solution to this problem was presented in [4] assuming the unit delay model, in which no delay is associated with any gate or interconnection within a cluster, and a delay of one time unit is encountered along any inter-cluster connection. A more realistic delay model called the general delay model was proposed in [5] in which each gate has an intrinsic delay value, no delay is associated with an interconnection within the same cluster, and finally a constant delay is associated with an inter-cluster connection. An optimal area-constrained clustering algorithm using this model was proposed in [6]. The clustering algorithm in [6] was then extended to consider pin constraints as well as area constraints [8].

Since most of today's circuit designs are very large in size, a multi-level clustering approach may result in reducing the complexity by a significant order. It is also important for mapping circuits onto hierarchical architectures like Altera's APEX FGPAs [1]. Cong et al. [2] developed a two-level clustering algorithm (TLC) and proved that the performance driven multi-level clustering problem is NP-hard. In their proposed heuristic algorithm a first-level cluster is defined as a set of nodes, and a second-level cluster is characterized as a set of first-level clusters, both with prescribed area bounds. TLC creates a two-level cluster for all nodes in the circuit and then covers the whole circuit based on the clusters. The cluster creation process constructs a second-level cluster and all first-level clusters within it at the same time. TLC is not easily extensible to solve circuit clustering problems with more than two levels due to the concurrent creation of second and first-level clusters. Moreover, TLC does not allow node duplication within a second-level cluster, which makes it less efficient, since allowing node duplication in second-level clusters may result in reducing the interconnections between clusters, which may reduce the circuit delay.

Recently, Sze et al. [7] presented a multi-level circuit clustering algorithm (MLC) that recursively clusters the circuit until the required number of clustering levels is achieved. At each level of clustering, a graph contraction algorithm is used to contract each cluster and build a new graph for the next level of clustering. Although MLC permits node duplication in all clustering levels, the final area of the circuit, measured as the number of clusters, is very high. Also, though not reported in [7], the actual area increase of the circuit (number of duplicated nodes) would be high as well. Our experiments show that the large number of clus-

ters results in slow placement process using the state-of-the-art placement tools.

To reduce the area overhead, we present a node duplication control strategy along with a cluster packing method embedded into a recursive clustering algorithm. Our clustering algorithm that is capable of clustering the circuit into any arbitrary number of levels (specified by the user) is a top-down clustering strategy based on the clustering algorithm in [6]. The algorithm creates clusters for each level separately from a desired level to the first level. At each level of the clustering the clusters generated from the previous level are transformed into smaller subcircuits and the clustering is recursively applied on each subcircuit. We are able to reduce the final circuit delay by allowing node duplication in all clustering levels. Using our node duplication control and cluster packing, which are carried out at each level of clustering, we can control the amount of area increase.

The rest of the paper is organized as follows: section 2 presents definitions and problem formulation. Our proposed algorithm is described in detail in section 3. Section 4 discusses experimental results, and finally, section 5 concludes the paper.

2. Definitions and Problem Formulation

A combinational Boolean network N can be represented as a directed acyclic graph $G = (V, E)$, where V is the set of nodes, and E is the set of directed edges. Each node, $v \in V$, represents a logic element (LE) equivalent to a 4-input lookup-table or 4-LUT, and each edge, $(u, v) \in E$, represents that node u is a fan-in of node v (u and $v \in V$). Let $input(v)$ and $output(v)$ be the set of all fan-in and all fan-out nodes of v, respectively. A primary input node is a node with fan-in 0, and a primary output node is a node with fan-out 0. Each node v has an area cost of $w(v)$.

Let n be the desired level of circuit clustering and network N can be represented by a single cluster at level $n+1$ defined as $\{C_1^{n+1}\}$ with the set of nodes $V_1^{n+1} = V$. An i-th-level cluster C^i, $n \geq i \geq 1$, is a set of nodes $V^i \subseteq V^{i+1}$ with an area constraint of M_i. The area constraint of each cluster C^i is defined as $W(C^i) = \sum_{u \in C^i} w(u) \leq M_i$. Let m_i be the number of i-th-level clusters and $m_{n+1} = 1$. Each cluster C_j^i, $(1 \leq i \leq n,\ 1 \leq j \leq m_i)$ can be represented by a subgraph $G_j^i(V_j^i, E_j^i)$ of G_k^{i+1}, for some $k \in \{1, \ldots, m_{i+1}\}$, induced by V_j^i and let $G_1^{n+1} = G$.

Each node $v \in V$ has an intrinsic delay $\delta(v)$ associated with it. Assuming constant delay values $D_n > D_{n-1} > \ldots > D_1 > D_0$, the delay of an edge connecting two nodes in different i-th-level clusters, but in the same $(i+1)$-th-level cluster is set to D_i, $(1 \leq i \leq n)$. More specifically, the inter-cluster delay at level i, $D^i(u, v)$, $1 \leq i \leq n$ is defined as $D^i(u, v) = D_k$ where k is the smallest in $\{i, \ldots, n\}$ such that both u and v are placed in one of the top-level clusters C^{k+1}. The delay of an edge within the same i-th-level and $(i+1)$-th-level clusters is set to D_0.

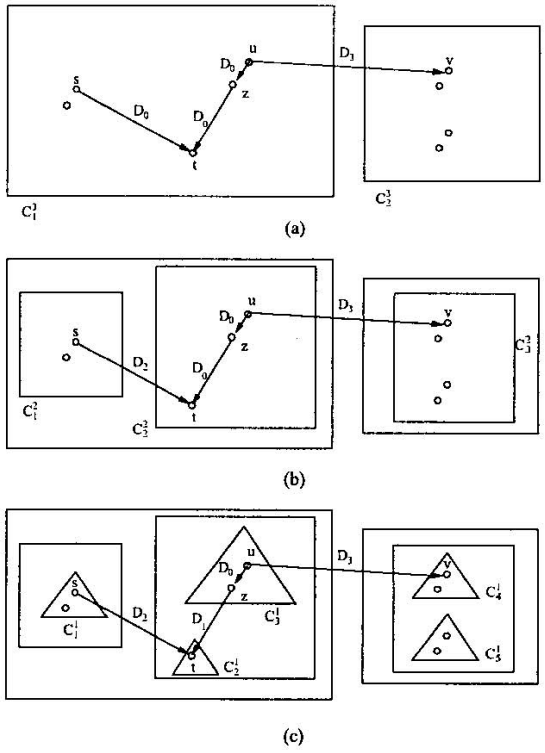

Figure 1. Assigning intra-cluster and inter-cluster delays to circuit connections during clustering (third-level, second-level and first-level clusters are illustrated by rectangles, squares, and triangles, respectively) (a) third level (b) second level (c) first level

Fig. 1 shows an example of assigning delay values $D_n, \ldots, D_0$, to circuit connections during clustering. Let the desired number of clustering levels be $n = 3$ and the area constraints at each level be $M_3 = 5$, $M_2 = 4$ and $M_1 = 2$ (so, the maximum number of nodes in the third, second and first-level clusters would be 5, 4 and 2 nodes, respectively). Nodes are shown as circles whereas third-level, second-level and first-level clusters

are illustrated as rectangles, squares, and triangles, respectively. Note that only some of the connections have been shown. Suppose that at the end of the first iteration of the clustering algorithm two third-level clusters C_1^3 and C_2^3 are generated (Fig. 1(a)). During clustering, all connections between these two clusters have a delay of D_3 (ex. (u,v)) and all intra-cluster edges (ex. (u,z), (z,t), (s,t)) have a delay of D_0. Assume that at the next level of clustering three second-level clusters are generated: C_1^2, C_2^2, and C_3^2 (Fig. 1(b)). At this level, since u, z and t are at the same second-level cluster, C_2^2, a delay of D_0 is incurred on (u,z) and (z,t) connections. However, the delay incurred on (s,t) would be D_2 since s and t are in different second-level clusters, but in the same third-level cluster. The delay on (u,v) is still D_3 since u and v are in different second-level and third-level clusters. Finally, if at the last level of clustering a total of five first-level clusters $(C_1^1,\ldots,C_5^1)$ are generated (Fig. 1(c)), as in the previous level, the delay incurred on (u,v) and (s,t) are D_3 and D_2, respectively; however, the delay of (z,t) would be D_1 since z and t are in the same second-level cluster (C_2^2); but in different first-level clusters.

The delay through a cluster is the maximum delay along any path from an input node of the cluster to an output node of the cluster. The delay along a path is the sum of the gate delays and edge delays on the path.

The recursive clustering problem with area constraints is then defined as follows: Given a combinational network N and a desired clustering level n, $n \geq 2$, divide the $(n+1)$-th-level cluster into a set of $\{C_1^n, C_2^n, \ldots, C_{m_n}^n\}$ of n-th-level clusters, and recursively divide all the i-th-level clusters into a set of $\{C_1^{i-1}, C_2^{i-1}, \ldots, C_{m_{i-1}}^{i-1}\}$ of $(i-1)$-th-level clusters until the set of first-level clusters is obtained, such that the delay of each cluster is minimized. The clusters of each level must satisfy the following conditions:[1]

$$\forall i \in \{n,\ldots,1\}, \forall j \in \{1,\ldots,m_i\}, \exists k \in \{1,\ldots,m_{i+1}\},$$

$$C_j^i \subseteq V_k^{i+1}, \qquad s.t. \begin{cases} W(C_j^i) \leq M_i, \\ \cup_{j=1}^{m_i} C_j^i = V \end{cases}$$

3. The RCP Algorithm

Fig. 2 shows our proposed clustering algorithm referred to as RCP (Recursive timing driven Clustering followed by cluster-Packing). The algorithm gets the graph G, the desired level of clustering n, maximum cluster sizes at each level $M_n,\ldots,M_1$, inter-cluster delays $D_n,\ldots D_1$, intra-cluster delay D_0, and maximum node slacks at each level $slack_{max}^n,\ldots,slack_{max}^1$. The clustering starts at the n-th-level and the single cluster C_1^{n+1} is transformed into the graph G_1^{n+1} which is equal to G. Then our updated single-level clustering algorithm is applied to G and returns the set of n-th-level clusters. After a post processing routine on the generated clusters (discussed below), the clustering at the n-th-level is finished and subsequent clustering levels are generated accordingly. At each i-th-level clustering, lines 4 to 7 are repeated m_{i+1} times, which is the number of $(i+1)$-th-level clusters generated at the previous level. Note that at each iteration, T^i contains the set of i-th-level clusters generated from clustering one of the $(i+1)$-th-level clusters. C^i, which contains all the i-th-level clusters, is obtained as the union of all T^i sets generated at each iteration.

In *Cluster_to_Netlist* procedure (line 4), at the n-th level, no transformation is done since C_1^{n+1} is the same as the original circuit. At the next level, the procedure transforms each n-th-level cluster C_j^n into a graph G_j^n which is the subgraph of G induced by V_j^n. At other levels of clustering the same transformation is done accordingly.

We now explain the *Post_Processing* procedure that is applied to each set of clusters generated from a single upper level cluster. Since node duplication is allowed at each level, it is possible that an already feasible cluster (in terms of area constraints) at a level i is no longer feasible after node duplication in the lower level i-1. Also note that we do not restrict the number of clusters inside an upper level cluster; however, there might be some limitation on this value. The *Post_Processing* procedure takes care of such problems by applying a recursive partitioning algorithm on the upper clusters. We leave the post processing procedure to the placement tool and let it decide if such processing is needed on the cluster. It should be noted that the number of clusters recorded in the experimental results are the number of feasible clusters in each level after placement.

Algorithm RCP (G, n, $M_n,\ldots,M_1$, $D_n,\ldots,D_1,D_0$, $slack_{max}^n,\ldots,slack_{max}^1$)
/* graph G is represented by a single cluster C_1^{n+1}. */
1. **for** $i = n$ **downto** 1 **do**
2. $\quad C^i = \emptyset$;
3. $\quad$ **for** all clusters $C_j^{i+1} \in \{C_1^{i+1}, C_2^{i+1}, \ldots, C_{m_{i+1}}^{i+1}\}$
4. $\quad\quad G_j^{i+1} \leftarrow$ Cluster_to_Netlist (C_j^{i+1});
5. $\quad\quad T^i \leftarrow$ Updated_Single_Level_Clustering $(i, G_j^{i+1}, M_i, D_n,\ldots,D_i,D_0, slack_{max}^i)$;
6. $\quad$ Post_Processing (T^i);
7. $\quad C^i \leftarrow C^i \cup T^i$;

Figure 2. RCP algorithm

[1] Note that V includes the duplicated nodes, if any.

3.1. Updated Single-Level Clustering

Fig. 3 shows the updated single-level clustering algorithm. The inputs to the algorithm are the current clustering level *level*, the netlist graph $\hat{G}$ that will be clustered, area constraint at this level M_{level} and the necessary constant delay values. *Labeling* (line 1) finds the best cluster rooted at each node v in $\hat{G}$. *Clustering* (line 2) constructs clusters starting from the output nodes using the clusters obtained in the labeling phase. After clustering, the number of clusters may be reduced using a cluster elimination algorithm. The last two steps are node duplication control and cluster packing that reduce the number of duplicated nodes and the number of clusters.

3.1.1. The Labeling and Clustering Phases. Let N_v, $v \in \hat{G}$, be the subgraph of the graph $\hat{G}$ induced by v and all of its predecessors, with v being the only output node. A cluster rooted at v, denoted by C_v, is a subgraph of N_v containing v. Let $l(v)$ denote the label (delay) of a node v.

In the labeling phase, we label each node v with the maximum delay at v in an optimal clustering of N_v. For each of the cluster input nodes v, we assign $l(v) = 0$. All other nodes are processed in topological order. To label a node v, all of whose predecessor nodes $u \in N_v$ have been labeled, we first compute $l'(u)$ using $l'(u) = l(u) + dist(u,v) * (D_0 + \delta)$, where $dist(u,v)$ is the maximum number of levels between u and the root node v. $l'(u)$ is actually the maximum total node and edge delay along any path in N_v from u to v. After calculating l' for all nodes in $N_v \backslash \{v\}$ we try to cluster v with as many nodes as the capacity constraint permits to make C_v. All nodes in $N_v \backslash \{v\}$ are sorted into a list P in order of decreasing value of l', so that nodes with greater l' values are selected first to be included in C_v.

After building C_v, v is labeled by considering all possible paths from an input to the output of v. Since no cluster input node is allowed to be in a cluster all the paths cross C_v. The label of node v is then computed using:

$$l(v) = l'(u) + D^{level}(u,v) - D_0$$

where u is either the last node in N_v which could not be included in C_v (due to capacity constraint) or the first cluster input node selected from P. Since P is sorted in order of decreasing value of l', the value of $l'(u)$ would be the maximum l' value among all nodes that are fan-in to C_v. Note that we need to deduct the intra-cluster delay D_0 for the edge between u and its sink inside C_v because it is now an inter-cluster edge whose delay is specified by $D^{level}(u,v)$.

Fig. 4 shows the labeling phase of a simple network with two primary input nodes a and b with intrinsic delays of zero. Let the capacity M of a cluster be 3, and assume a single-level clustering is being done. Let the node delay δ, the intra-cluster delay D_0, and the inter-cluster delay D_1 be 0.5, 1.0, and 2.0, respectively. The labels of nodes c, e, and f can be computed as shown in Figs. 4(a)-4(c) (node d is labelled like c). For example, in Fig. 4(b) after finding the l' values of the nodes in N_e and sorting them into the list P we have: $P = \{c, d, a, b\}$. Each cluster can contain up to three nodes, so the final cluster for e would be $C_e = \{e, c, d\}$. The label of e is then computed using $l(e) = l'(a) + D_1 - D_0 = 3 + 2 - 1 = 4$. An important point when sorting the nodes in P is that if two or more nodes have the same l' value, they must be sorted in decreasing order of their l value. This is done to prevent the cluster from being made of two (or more) distinct groups of nodes. For example, after labeling nodes in N_f (Fig. 4(c)), nodes f and any two nodes among e, d, and c can be in C_f (because all have equal l' values). So the cluster could be $C_f = \{f, c, d\}$ (Fig. 5(a)), which consists of two disjoint groups of nodes. Clearly, $\{f, e, d\}$ or $\{f, e, c\}$ are more acceptable (Figs. 5(b) and 5(c)). By sorting nodes (with equal l' values) based on their l values, node e and one of the nodes c or d would be selected to be in C_f.

The clustering phase generates the necessary clusters to form the optimal clustering solution at this level. First, the clusters for the cluster output nodes are generated and then for each input node u of the generated clusters, the cluster rooted at u is added to the set of final clusters. For example, for the simple network of Fig. 4 $C_f = \{f, e, c\}$ and $C_d = \{d\}$ would be the final clusters of the network. After clustering, we use a rooted cluster elimination algorithm, presented by Yang et al. [8], to eliminate as many rooted clusters as possible by substituting the roots (i.e. the only output node) of these clusters with their copies in other clusters provided that the computed delay of the clustered circuit is not increased. Note that if a cluster is elimi-

Algorithm Updated_Single_Level_Clustering ($level$, $\hat{G}$, M_{level}, $D_n, \ldots, D_{level}, D_0$, $slack_{max}^{level}$)
1. initial_clusters_set ← Labeling ($level$, $\hat{G}$, M_{level}, $D_n, \ldots, D_{level}, D_0$);
2. cluster_set ← Clustering ($\hat{G}$, initial_clusters_set);
3. Rooted_Cluster_Elimination (clusters_set);
4. Node_Dup_Control ($level$, $\hat{G}$, clusters_set, $slack_{max}^{level}$);
5. packed_set ← Cluster_Packing (cluster_set, M_{level});

Figure 3. Updated single-level clustering algorithm

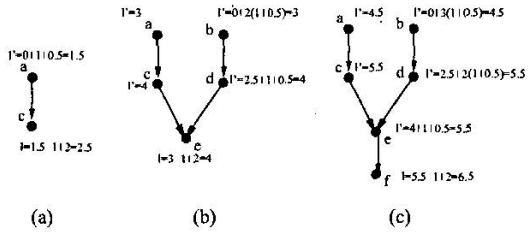

Figure 4. The labeling phase for subgraph N_f. (a) labeling c (b) labeling e (c) labeling f

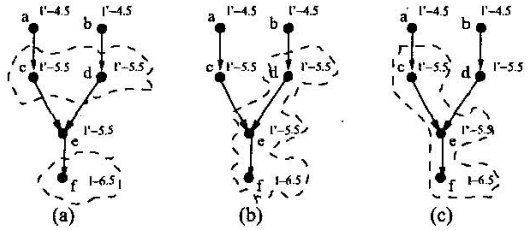

Figure 5. Possible clusters for node f (a) $C_f = \{f, c, d\}$ (b) $C_f = \{f, e, d\}$ (c) $C_f = \{f, e, c\}$

nated, the copy of its root in the other cluster becomes an output node of that cluster.

3.1.2. Node Duplication Control and Cluster Packing. Since the number of duplicated nodes may be large, we impose some restrictions on the amount of node duplication. We control node duplication based on the node slack —nodes with slacks greater than a predefined value are not duplicated. First the labels of all nodes are recalculated. Note that we should not use old node labels, because old labels for each node are generated based on the cluster rooted at that node. However, some clusters may not exist in the final solution either because they are not included in clustering algorithm or because they have been eliminated during rooted cluster elimination. So we need to calculate new labels based on the existing clusters. To find node slacks we first calculate the final labels of all nodes in the circuit based on the current clustering results. The label of each input node of a cluster is set to 0. Then the label of any other node v can be calculated as $l(v) = \max_{u \in input(v)} l(u) + \delta(v)$ where

$$l(u) = \begin{cases} l(u) + D_0 & (u,v) \text{ is an intra-cluster edge} \\ l(u) + D^{level}(u,v) & (u,v) \text{ is an inter-cluster edge} \end{cases}$$

After calculating the labels, the *required labels* of all nodes are computed. We first set the required label of all cluster output nodes to the maximum label among all such nodes. Then, the required label of any other node v, can be calculated as $rl(v) = \min_{u \in output(v)} rl(u) - \delta(u)$ where

$$rl(u) = \begin{cases} rl(u) - D_0 & (v,u) \text{ is an intra-cluster edge} \\ rl(u) - D^{level}(v,u) & (v,u) \text{ is an inter-cluster edge} \end{cases}$$

Finally, we calculate the slack of each node v as $slack(v) = rl(v) - l(v)$.

After calculating node slacks, all nodes u in each cluster in the final cluster list are processed in reverse topological order. First we check if a node u is duplicated and if not, we just keep the node in the cluster. Otherwise, we check if u is an output node in any cluster. If u is not an output node in any cluster we keep the node in the clusters where u can be duplicated (the slack value is less than the maximum). If u can be duplicated, but none of its fan-outs exists in that cluster, we remove u —this can be done since the cluster nodes are processed in reverse topological order. If u is an output node in some cluster we keep u in the cluster in which it is an output node and we check for the duplication possibility in other clusters. In this way, since output nodes of a cluster have the best delay among all other duplicates of that node, we always keep such nodes and check for duplication permission for other copies of the nodes.

To reduce the number of clusters at each level we use the generalized packing algorithm proposed in [3] with the addition of imposing pin constraint on the number of inputs for the first-level clusters. This is done to comply with architectural constraints in our experimental tests and is discussed in section IV.

3.2. Algorithm Complexity Analysis

The updated single level clustering algorithm takes $O(|V|^2 log|V| + |V||E|)$ time. This is actually the time complexity for the labeling algorithm, which can be computed with the same analysis as in [6]. The other steps in the updated single level clustering algorithm have lower complexities than the labeling: in node duplication control, reverse topological ordering of the nodes for a cluster C_j^i takes $O(M_i + |E_j^i|)$; thus the total duplication control time would

be $O(m_i(M_i + |E^i|)) \leq O(|V|(|V| + |E|))$. Cluster packing takes $O((\max_{i=1,\ldots,n}(m_i))^2) \leq O(|V|^2)$ [3] (m_i is the number of clusters at level i). Therefore, for the multi-level circuit clustering problem with n levels, the overall time complexity of the RCP algorithm is $O(n(|V|^2 log|V| + |V||E|))$.

4. Experimental Results

We have implemented our RCP algorithm in C++ and ran our experiments on a Pentium III 900 MHz CPU with 512 MB of memory and compared our algorithm to the TLC implementations which are obtained from the authors of [2]. We used the commercial synthesis tool Quartus II v.2.0 from Altera to test our results. As in [2], we used the EP20K600EFC672-1X device from the APEX20KE family. We tested our algorithm for the two-level hierarchy of APEX architecture in which 10 4-LUTs are grouped together and make a *logic array block* (LAB) and then 16 such LABs are grouped together and make a MegaLAB. We assigned the second-level clusters to MegaLABs ($M_2 = 160$) and the first-level clusters to LABs ($M_1 = 10$). The delay parameters are the same as the ones used in [2]: $D_2 = 1.57ns$, $D_1 = 0.85ns$, $D_0 = 0.36ns$, and for all nodes v, $\delta(v) = 0.61ns$ and $w(v) = 1$.

We used combinational circuits from the MCNC benchmarks that were used in [2]. Our algorithm generates the set of clusters along with the new netlist of the circuit that contains duplicated nodes (if any) and their corresponding duplicated nets. We used the LogicLock feature provided by Quartus and assigned each of our clusters to one LogicLock region. LogicLock region information is kept in a specific file with an ESF specification that is generated by our algorithm. It should be noted that clusters in [2] are represented using Clique-based clusters in ESF files, however, since Quartus II v.2.0 no longer supports MegaLAB size Cliques, we converted their clustering information from Clique-based to LogicLock-based by changing the syntax of the ESF files. Also, as in [2], we imposed a constraint on the maximum number of inputs that a first-level cluster (that corresponds to a LAB) can have since LABs in APEX architecture cannot have more than 22 inputs.

We tested several metrics for our cluster packing as in [3]. The best delay result was achieved when clusters were merged using common inter-cluster net packing in both clustering levels. Table 1 illustrates the delay results for TLC implementations and our algorithm when common inter-cluster net packing is used in both levels. The second column shows the results for TLC when no node duplication (No ND) is allowed; columns 3-4 show the results for TLC with full node duplication (Full ND), and columns 5-6 and 7-8 show the results of our algorithm when no node duplication is allowed in any level, and when full node duplication is allowed in both levels, respectively. The "delay" column represents the delay of each clustered circuit in ns and the "% dec" column lists the percentage of delay reduction compared with the TLC No ND implementation. The results demonstrate that our algorithm with Full ND achieves, on average, 7.3% more delay reduction compared with the best results achieved by TLC. An interesting result is that our algorithm in No ND mode has better delay results than the TLC Full ND implementation. This shows the effectiveness of our clustering algorithm even when there is no node duplication. We could not compare our results with MLC [7] since it does not produce any circuit or clustering information, but rather it just reports the estimated delay based on the generated clusters.

Table 2 shows the area (number of second-level clusters, NC, and the number of LEs, LUT) and FPGA compile time (in $mm:ss$) corresponding to the results of Table 1. For the No ND modes (columns 2-3 for TLC and 8-9 for our algorithm) there is no increase in the number of LEs since no duplication is allowed. For the Full ND modes (columns 4-7 for TLC and 10-13 for our algorithm) the $\%inc$ columns list the percentage of LUT increase compared with the original circuits. The FPGA compile time columns list the total compile time (Synthesis, Fitting, and Timing Analysis) reported by Quartus. Compared with TLC Full ND, RCP Full ND reduces FPGA compile time by 36%. This reduction is mainly due to the small number of second-level clusters. So, even though our Full ND algorithm results in a 20% LUT increase compared with the TLC Full ND implementation, it can reduce both the average delay and compile time results. It should be noted that TLC does not contain cluster packing and just merges very small clusters to reduce the number of clusters.

Table 3 shows the effect of our node duplication control strategy (RCP Part. ND). The results are for the case when nodes are not duplicated if their slack is greater than $2ns$ and $0ns$, in second-level clusters and first-level clusters, respectively ($slack^2_{max} = 2ns$, $slack^1_{max} = 0ns$). As shown in row 1, the average LUT overhead is decreased by 68% and the average FPGA compile time is also reduced by 50%. These area and compile time reductions are achieved at the expense of a 4.2% decrease in the the average delay improvement. TLC has another implementation with partial node duplication (TLC Part. ND). Although this implementation reduces the average LUT increase more than RCP Part. ND, however, the average delay de-

Table 1. Delay comparison

	TLC (No ND)	TLC (Full ND)		RCP (No ND/No ND)		RCP (Full ND/Full ND)	
Circuit	delay	delay	% dec	delay	% dec	delay	% dec
alu	36.97	33.54	9.3	29.21	21.0	25.59	30.8
apex2	33.98	32.05	5.7	31.52	7.2	29.58	12.9
apex6	18.60	20.91	-12.4	19.10	-2.7	17.52	5.8
C1908	24.96	25.17	-0.8	25.91	-3.8	24.03	3.7
C5315	32.51	31.75	2.3	30.53	6.1	27.68	14.9
C880	24.62	20.91	15.1	26.45	-7.4	22.36	9.2
dalu	22.41	22.05	1.6	22.40	0.0	20.58	8.2
des	28.94	29.42	-1.7	27.18	6.1	24.10	16.7
i10	46.21	44.10	4.6	41.64	9.9	42.64	7.7
i9	21.52	21.82	-1.4	23.62	-9.8	20.57	4.4
k2	27.65	26.54	4.0	28.93	-4.6	22.75	17.7
large	31.04	30.25	2.5	28.97	6.7	27.37	11.8
misex3	27.47	29.61	-7.8	20.71	24.6	28.23	-2.8
too_large	19.10	20.27	-6.1	18.26	4.4	18.18	4.8
vda	22.55	18.53	17.8	22.12	1.9	19.65	12.9
x3	20.13	17.05	15.3	18.89	6.2	19.05	5.4
Average			3.0		4.1		10.3

Table 2. Area and FPGA compile time comparisons

	TLC (No ND)		TLC (Full ND)				RCP (No ND/No ND)		RCP (Full ND/Full ND)			
Circuit	area	FPGA compile time	area			FPGA compile time	area	FPGA compile time	area			FPGA compile time
	NC		*LUT*	% inc	*NC*		*NC*		*LUT*	% inc	*NC*	
alu	26	01:27	1431	345.8	86	03:38	5	01:15	2350	632.1	16	03:58
apex2	215	10:57	3744	99.4	267	21:12	21	04:51	3425	82.4	37	08:53
apex6	99	01:55	516	31.3	99	02:03	6	01:25	493	25.4	5	01:35
C1908	25	01:12	741	411.0	25	02:29	2	01:00	656	352.4	5	02:02
C5315	120	04:04	2366	281.6	154	14:01	12	02:24	2329	275.6	25	06:46
C880	25	01:19	431	147.7	26	01:45	4	01:05	485	178.7	5	01:35
dalu	16	01:46	829	65.8	16	02:21	6	01:37	882	76.4	8	02:16
des	244	10:24	4268	168.3	245	29:23	17	04:15	3164	98.9	29	09:59
i10	203	08:06	3527	254.5	276	40:59	13	03:23	6907	594.2	55	37:29
i9	63	02:06	521	38.6	63	02:12	5	01:27	603	60.4	6	01:31
k2	32	02:29	2109	308.7	106	06:39	7	02:06	2254	336.8	20	04:51
large	129	05:08	1897	105.7	172	07:57	12	02:52	2023	119.4	17	04:16
misex3	100	05:27	2458	75.9	149	08:38	15	03:18	2809	101.1	28	05:47
too_large	3	01:04	256	36.9	3	01:09	2	01:08	295	57.8	3	01:11
vda	33	01:39	1128	287.6	38	02:43	4	01:31	937	222.0	8	02:07
x3	99	01:59	484	28.4	99	01:59	5	01:35	490	30.0	5	01:30
Average	89.5	03:49		168.0	114.0	09:19	8.5	02:12		202.7	17.0	05:59
Total	1432	1:01:02			1824	2:29:08	136	0:35:12			272	1:35:46

Table 3. Partial node duplication compared with RCP Full ND

	Avg *LUT* dec %	Total compile time dec %	%inc in Avg delay
RCP (Part. ND/ Part. ND)	68	50	4.2
TLC (Part. ND)	82	15	9.7

crease and compile time are worse than RCP.

5. Summary

We have presented an effective recursive algorithm for circuit clustering for delay minimization. A node removal algorithm and a cluster packing algorithm were also presented and used to reduce the amount of node duplication and the number of clusters at each level of clustering. Experimental results for the two-level clustering problem using Quartus II v2.0 showed that our algorithm achieved, on average, 7.3% more delay reduction compared with the best results in [2]. Also the total FPGA compile time reported by Quartus was reduced by 36%. To decrease the area overhead resulted in our algorithm, a partial node duplication strategy was proposed that resulted in 68% reduction in area overhead and 50% reduction in the average FPGA compile time. Currently, we are working on our partial duplication algorithm to improve the delay results.

Acknowledgements

The authors would like to thank Dr. Jason Cong and Michail Romesis for providing their TLC implementa-

tions, and Chin-Ngai Sze and Dr. Ting-Chi Wang for providing the MLC binary code.

References

[1] Altera Corp., *http://www.altera.com/products/devices/apex/features/apx-architecture.html.*

[2] J. Cong and M. Romesis, "Performance-Driven Multi-Level Clustering with Application to Hierarchical FPGA Mapping", *Design Automation Conference*, pp. 389-394, 2001.

[3] M. Eslami Dehkordi and S. D. Brown, "The Effect of Cluster Packing and Node Duplication Control in Delay Driven Clustering", *IEEE Intl. Conf. on Field-Programmable Technology*, pp. 227-233, 2002.

[4] E.L. Lawler, K.N. Levitt, and J. Turner, "Module Clustering to Minimize Delay in Digital Networks", *IEEE Trans. on Computers*, Vol. C-18, No. 1, pp. 47-57, 1969.

[5] R. Murgai, R.K. Brayton, and A. Sangiavanni-Vincentelli, "On Clustering for Minimum Delay/Area", *Intl. Conference on Computer-Aided Design*, pp. 6-9, 1991.

[6] R. Rajaraman and D.F. Wong, "Optimal Clustering for Delay Minimization", *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, Vol. 14, No. 12, pp. 1490-1495, 1995.

[7] C.N. Sze and T.-C. Wang, "Performance-Driven Multi-Level Clustering for Combinational Circuits", *ASP-DAC*, pp. 729-734, 2003.

[8] H. H. Yang and D. F. Wong, "Circuit Clustering for Delay Minimization Under Area and Pin Constraints", *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, Vol. 16, No. 9, pp. 976-986, 1997.