Performance-Driven Multi-Level Clustering with Application to Hierarchical FPGA Mapping

Jason Cong
UCLA Computer Science Department
Los Angeles, CA 90095
cong@cs.ucla.edu

Michail Romesis
UCLA Computer Science Department
Los Angeles, CA 90095
michail@cs.ucla.edu

ABSTRACT

In this paper, we study the problem of performance-driven multi-level circuit clustering with application to hierarchical FPGA designs. We first show that the performance-driven multi-level clustering problem is NP-hard (in contrast to the fact that single-level performance-driven clustering can be solved in polynomial time optimally). Then, we present an efficient heuristic for two-level clustering for delay minimization. It can also provide area-delay trade-off by controlling the amount of node duplication. The algorithm is applied to Altera's latest APEX FPGA architecture which has a two-level hierarchy. Experimental results with combinational circuits show that with our performance-driven two-level clustering solution we can improve the circuit performance produced by the Quartus Design System from Altera by an average of 15% for APEX devices measured in terms of delay after final layout. To our knowledge this is the first in-depth study for the performance-driven multi-level circuit clustering problem.

1. INTRODUCTION

Circuit clustering is a technique that groups the gates of a circuit into clusters under the area bound and/or pin constraints to optimize certain metrics. Commonly used metrics include maximization of the connectivity within clusters (e.g.[6,10,16]) or minimization of the delay of the clustered circuits (e.g. [2,4]). In this paper, we focus on delay minimization of the clustered circuit. Circuit clustering is an important technique for various reasons. First, all modern circuit designs are very large in size. Clustering can reduce the complexity by a significant factor. Second, clustering can improve the quality of the results of other operations (such as partitioning or placement) especially in the multi-level optimization framework. For example, the hMetis algorithm used the multi-level clustering scheme for cutsize minimization in partitioning [11]. PRIME [4] and HPM [2] use simultaneous circuit partitioning/clustering with retiming for performance optimization. Finally, multi-level clustering is important for mapping circuits onto hierarchical architectures as shown in the next paragraph.

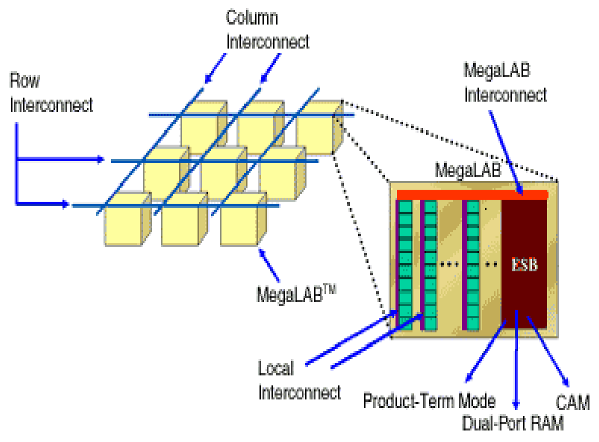

Figure 1 : APEX 20K MegaLAB Structure (Source: Altera Corp.)

An example of the increasing usage of hierarchical architectures can be seen in the FPGA field. In 1999, Altera shipped its APEX20K devices with up to 51,840 logic elements (equivalent to 4-input lookup-tables or 4-LUTs) and 60,000 to 1.5 million usable gates [3]. In order to cope with the complexity of such high-capacity devices Altera uses a two-level architecture (Figure 1). The first-level cluster is called a *logic array block* (*LAB*) consisting of 10 4-LUTs connected by the local interconnect array. Sixteen such LABs form the second-level cluster, called a *MegaLAB*. This architecture can take advantage of the locality of interconnections and use the faster local and semi-global interconnections to improve the performance of the circuit.

Previous work on performance-driven clustering focused only on the single-level clustering formulation. The early work by Lawler et al. [12] presented a polynomial-time delay-optimal algorithm for area-constrained circuit clustering under the unit delay model. In this model, a constant delay is associated with every interconnection between two gates in different clusters, and no delay is associated to an interconnection within the same cluster. A more realistic model, called the "general delay model", was proposed by Murgai et al. [13], in which each gate may have a different delay. It assumes no delay for any interconnection inside the cluster and a constant delay for every interconnection between the clusters. Rajaraman and Wong [15] presented the first delay-optimal algorithm for area-constrained clustering under the general delay model. Other algorithms considering both area and pin constraints have been developed [17]. All the results discussed so far

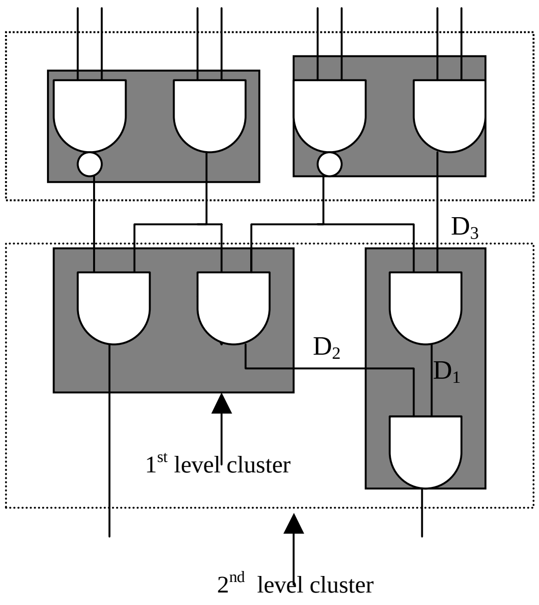

Figure 2 : An example of a Two-Level Clustering Solution

apply to combinational circuits only. For sequential circuits, Pan et al. [14] proposed a polynomial-time clustering algorithm with retiming that achieves quasi-optimal delay under the general delay model. PRIME [4] provides significant space and time complexity improvement of [14] while maintaining quasi-optimal delay solutions.

It is obvious that efficient performance-driven multi-level clustering is important for hierarchical architectures with different delays among the components at different levels of the chip. In this paper, we study the problem of performance-driven multi-level circuit clustering for combinational circuits. We first show that the performance-driven multi-level clustering problem is NP-hard (in contrast to the fact that single-level performance-driven clustering can be optimally solved in polynomial time). Then, we present an efficient heuristic for two-level clustering for delay minimization. It can also provide the area-delay trade-off by controlling the amount of node duplication. The algorithm is applied to the latest APEX FPGA architecture from Altera which has a two-level hierarchy. Experimental results show that with our performance-driven two-level clustering solution we can improve the circuit performance produced by the Quartus Design System from Altera by an average of 15% for APEX devices measured in terms of delay after final layout.

The rest of the paper is organized as follows. Section 2 defines the circuit clustering problem under a two-level delay model. Section 3 discusses the complexity of the problem. Section 4 presents our heuristic algorithm, called *Two-Level Clustering* (TLC) for performance-driven two-level clustering. We discuss the two phases of the TLC algorithm, labeling and clustering and the area-delay trade-off by node duplication. Section 5 presents the experimental results and Section 6 concludes the paper.

2. PROBLEM FORMULATION

A combinational network can be represented as a directed acyclic graph $N= (V, E)$, where V is the set of nodes, and E is the set of directed edges. Each node in V represents a gate in the network and an edge (u,v) is in E if and only if there is an interconnection between gates u and v in the network. Every node u is associated with two parameters: delay $d(u)$ and area $w(u)$. A *first-level cluster* is a set of nodes $U \subset V$ of the network whose total area does not exceed a prescribed bound, say M_1. For a first-level cluster p, if we use $w(p)$ to denote the total area of the cluster, we have:

$$w(p) = \sum_{u \in p} w(u) \le M_1$$

A *second-level cluster* is a set of first-level clusters whose total area does not exceed another constant, M_2. For a second-level cluster b, if $w(b)$ denotes the area of b, we have :

$$w(b) = \sum_{p \in b} w(p) \le M_2$$

The two-level delay model that we use is the following:
1) Every node u has a delay $d(u)$.
2) An interconnection between two nodes in the same first-level cluster has a fixed delay D_1.
3) An interconnection between two nodes in different first-level clusters, but in the same second-level cluster has a fixed delay of D_2.
4) An interconnection between two nodes in different second-level clusters has a fixed delay D_3.

We can now define the two-level clustering problem as follows: Given a combinational network, cover the network with two-level clusters subject to the area constraints. A feasible solution to the two-level clustering problem contains two sets:

1) a set $S_1=\{V_1, V_2, ..V_n\}$ representing first-level clusters where:

$$V_i \subset V, w(V_i) \le M_1 \ \ for\ 1 \le i \le n\ \ and\ \bigcup_{i=1}^{n} V_i = V$$

2) a set $S_2=\{X_1, X_2, .., X_m\}$ representing second-level clusters where:

$$X_i \subset S_1, w(X_i) \le M_2 \ \ for\ 1 \le i \le m\ \ and\ \bigcup_{i=1}^{m} X_i = S_1$$

The clusters may have common nodes, but the clustered network must be logically equivalent to the input network. Our goal is to minimize the delay through the network according to the two-level delay model. The delay through the network is the maximum delay along any path from a primary input node to a primary output node (recall that we are dealing with combinational circuits). Figure 2 shows an instance of the two-level clustering problem where $M_1=2$ and $M_2=4$. For $d=1, D_1=2, D_2=3, D_3=4$ the circuit has a delay of 18.

In the next section, we discuss the complexity of the two-level clustering problem. In Section 4, we present a heuristic for this problem.

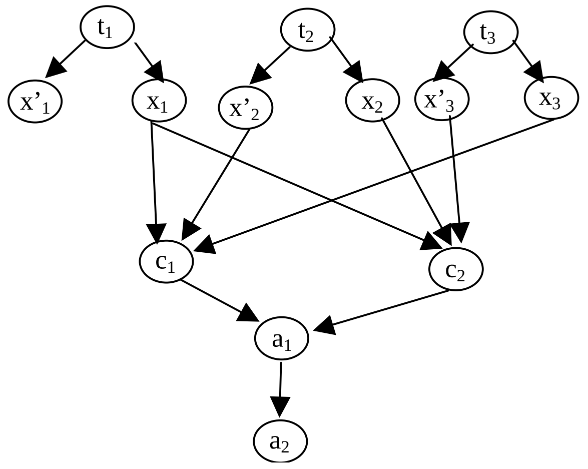

Figure 3 : An example of the graph for $c_1=x_1x'_2x_3$ and $c_2=x_1x_2x'_3$

3. PROBLEM COMPLEXITY ANALYSIS

In [15] the authors proved that the area-constrained single-level clustering problem under the delay model defined in Section 2 (without D_3) can be solved in polynomial time. In contrast, we show in this paper that when the problem is extended to two levels of clusters, it becomes NP-hard. In [8] we provide a proof for the NP-hardness of the two-level clustering problem. In this section, we outline the transformation procedure so that the reader can understand where the difficulty comes from. Generally speaking, the main difference between the single-level and the multi-level clustering problems is node duplication. In the single-level case, node duplication is performed whenever it may reduce the delay without consideration of the increase in the number of clusters. On the other hand, in the multi-level case node duplications can occur in the same second-level cluster (but among different first-level clusters). In this case, the cluster capacity constraint must be taken into consideration. For that reason in our heuristic we do not allow node duplication inside a second-level cluster for area-reduction reasons.

The decision version of the two-level clustering problem is the following: Given a combinational network, is there a feasible clustering solution such that the maximum delay of the network under the two-level delay model is smaller than a given constant *D*? In order to show that this problem is NP-hard we reduce the 3-SAT problem to it. It is known that the 3-SAT problem is NP-complete [9].

Given an instance of the 3-SAT problem with *V* variables and *C* clauses we make the transformation to the two-level clustering problem as follows: The network is represented as a graph with four types of nodes: literal nodes, clause nodes, consistency nodes and auxiliary nodes. There is a literal node corresponding to each of the literals x_i or x'_i in the 3-SAT problem. There is also a clause node c_j corresponding to each clause of the 3-SAT. For each variable x_i of the 3-SAT there is a consistency node t_i. We have also some auxiliary nodes. The weights of these nodes are given in [8].

The edges of the graph are defined as follows: For every literal node there is an edge from the corresponding consistency node to the literal node. For every clause node there are three edges from every literal node of the clause to the clause node. Also from every clause node there is an edge to the same auxiliary node. The auxiliary nodes create a chain. An example of the graph created can be seen in Figure 3. The groupings of the consistency nodes with the literal nodes correspond to the assignments to a variable in the 3-SAT instance. In [8] we prove that the maximum delay of the network can have a certain minimum value if and only if a satisfiable assignment exists.

4. THE TLC ALGORITHM

In this Section we present our algorithm, named *two-level clustering* (TLC), which works in two phases. In the first phase (labeling phase) we label each node *u* with an estimated maximum delay in our clustering result. During this phase we create for each node *u* a two-level cluster that has *u* as a root node. The nodes are visited in topological order.

In the second phase (clustering phase) we cover the network with the clusters created in the previous phase. In this phase the nodes are visited in reverse topological order.

4.1 Labeling Phase

In this phase we use the dynamic programming technique to compute the label *l(u)* for each node *u* from the primary inputs (PIs) to the primary outputs (POs) of the network in topological order. The label *l(u)* stands for the delay of node *u* in the two-level clustering solution computed by the TLC algorithm. For each of the primary inputs of the network, we assign *l(u)=d(u).* During the process we are visiting the predecessors of *u* until we fill a second-level cluster rooted at *u*.

We maintain two lists of candidate nodes to include in the cluster. The first is called a first-level list that includes the candidates for filling the current first-level cluster. This list contains all the fanins to the set of nodes included in the current first-level cluster. There is also a second-level list that includes the candidates for root nodes for the next first-level cluster. This list in the same way includes all the fanins to the set of nodes contained in the second-level cluster so far. The procedure is the following: From the second-level list we choose the root node for the next first-level cluster. Then we fill the first-level cluster by choosing nodes from the first-level list which is updated dynamically. This procedure is repeated until we fill the second-level cluster. We would also like to mention that we impose a constraint on the maximum number of inputs that a first-level cluster can have when applied to the APEX devices, because the first-level clusters correspond to LABs in the Altera devices which cannot have more than 22 inputs.

In order to choose what nodes to include in the cluster rooted at *u*, we use their label delay and their maximum distance calculated so far from the root node. For every node *v*, we use *g(v)* to denote the immediate successor of *v* with the maximum distance to the root node *u.* We define:

$$f(v,u)=l(v) + distance(g(v), u)$$

Obviously, *f(v,u)* is a lower bound on the delay along any path from a primary input to the root node *u* that passes through *v.* The greater the value of the function, the bigger the need to include the node in the cluster. Therefore we use *f(v,u)* as a guide for the choice of a node from the first-level list and the second-level list.

After we have finished filling the second-level cluster, we compute the label of the root node *u* of the corresponding second-level cluster as follows. This procedure is similar to the approach in [15] where they consider all possible paths from an input to the root node.

All paths can be divided into two categories:
a) *Paths that lie enitrely inside the second-level cluster*. Such paths start from a primary input node that is included in the second-level cluster. The maximum delay along any such path is :

$l_1(u)= max\{f(v,u)+D|v \in cluster(u) \cap PI, D=D_1$ if v and $g(v)$ are in the same first-level cluster else $D=D_2\}$

b) *Paths that cross the second-level cluster*. Among these paths the maximum delay is:

$l_2(u)= max\{f(v,u)+D_3|\ v$ is connected to at least one node of $cluster(u)$ but does not belong to $cluster(u)\}$

Then the label of *u* is the maximum of $l_1(u)$ and $l_2(u)$.

A description for the labeling phase is given in Fig. 4. In order to reduce the total area of the circuit, we do not allow any node duplications inside a second-level cluster. The reason is that it is not clear if such node duplication helps to reduce the delay, as it may reduce the delay of some first-level cluster,

```
LABEL(u):
distance(u)←d(u)
Second_Level_List←{u}
While second_level_cluster_area<$M_2$ AND
 Second_Level_List ≠∅
    Create new first-level cluster
    Choose from Second_Level_List node v with
        max(f(v,u))
    root_node ← v
    First_Level_List←{root_node}
    FILL_FIRST_LEVEL_CLUSTER(First_Level_List)
    Update Second_Level_List
    First_Level_List←∅
Endwhile
Compute $l_1$(u)
Compute $l_2$(u)
L(u)=max{$l_1$(u),$l_2$(u)}

* * * * * * * * * * * * * * * * * * * * * * * * * * * * *
FILL_FIRST_LEVEL_CLUSTER (First_Level_
List) :
While first_level_cluster_area<$M_1$ AND
 First_Level_List ≠∅
     Choose from First_Level_List node v with
         max(f(v,u))
     current_node ← v
     Add current_node to first-level cluster
     For every fanin w of current_node
         Add node w to First_Level_List
         Update g(w)
    Endfor
Endwhile
```

Figure 4 : The labeling phase of the TLC algorithm

but also reduce the capacity of the second-level cluster. Still we allow node duplications in different second-level clusters.

4.2 Clustering Phase

In the clustering phase we choose what clusters from those created in the previous phase to include in order to cover the network.

This phase is also similar to the clustering phase of [15]. We maintain a list L of nodes whose clusters we will include. Initially L contains all the primary outputs of the network. At each step we choose one node from L, we generate the cluster rooted from the node and we insert into the list all the inputs to that cluster. This way we can have node duplication as mentioned before. When L becomes empty, the generated clusters cover the whole network. Figure 5 shows the summary for the entire algorithm.

4.3 Area-Delay Trade-Off

Although our algorithm is performance-driven we may impose some restrictions on the amount of node duplications in order to control the area of the resulting solution. For this reason in the clustering phase we choose to duplicate only the nodes that belong to the ε-network of the circuit. A node belongs to the ε-network if its slack is smaller than a predefined value ε. The slack *s(u)* of a node *u* is computed as follows:

$$s(u)=q(u)-l(u)$$

where *l(u)* is the label of the node and *q(u)* is the required time of the node defined as:

$$q(u)=min\{q(v)-d(e)-d(u)|e(v,u)\in E\}$$

The timing slack is used to determine the timing criticality of a node. If we duplicate only the nodes of the ε-network we can have a big improvement in area traded for a small degradation of performance.

4.4 Algorithm Complexity Analysis

In order to make the complexity analysis of the algorithm we assume that the nodes have integer areas. We label each node of the network, so if the network has N nodes we repeat the labeling phase O(*N*) times. Since we want to fill the second-level cluster we will choose a new node O(M_2) times. Every time we choose a node we have to make updates for each fanin of that node. If the maximum in-degree of the network is

```
ALGORITHM: TLC
Sort the nodes in topological order
For every node u in network
        LABEL(u)
L←PO
While L is not empty
        Remove a node u from L
         Add cluster(u) to solution while duplicating only
the nodes belonging to ε-network
         Add all inputs of cluster(u) to L
Endwhile
```

Figure 5 : The TLC algorithm

```
Script.rugged;
Tech_decomp –a 1000 –o 1000;
Dmig –k 2;
Flowmap –k 4;
Greedy_pack –k 4;
```

Figure 6: Script for the LUT mapping

I, the whole labeling phase takes O($N{\cdot}M_2{\cdot}I$) time. In the case of FPGA devices where all nodes are 4-input LUTs, I is equal to 4. In this case the complexity of of the labeling phase becomes O($N{\cdot}M_2$). The clustering phase has an O(N) time compexity, so the total time complexity of the algorithm is O($N{\cdot}M_2{\cdot}I$).

Since we want to keep the information about every node's corresponding cluster, the space complexity of the algorithm is O($N{\cdot}M_2$).

5.EXPERIMENTAL RESULTS

We have implemented the TLC algorithm in C++/STL and integrated it into the UCLA RASP System [7]. We ran our experiments on a SUN ULTRA10 workstation with a 440 MHz CPU and 1024 MB of memory. The experimental procedure was the following: From a given gate-level netlist, we first ran a script, shown in Figure 6, for the UC Berkeley SIS System [1] and the UCLA RASP System including the FlowMap algorithm [5], to generate a 4-input LUT network. As an output we have two files: one with a .TDF extension describing the new network which is logically equivalent to the original network and a file with an .ESF extension that describes the clustering constraints. The first-level clusters are assigned to LABs (see Section 1) and the second-level clusters to MegaLABs.

We used the commercial synthesis tool Quartus II v.1.0 from Altera to test our results. For every circuit we ran four experiments. First we provided to Quartus as an input the original bounded network without any clustering constraints. Then we requested from Quartus to use the clustering results from our algorithm. We have three versions of our algorithm. The first one allows duplications without any restrictions at all. The second version duplicates only the nodes belonging to the ε-network of the circuit. The third version allows no duplication at all. The experimental flow is graphically presented in Figure 7.

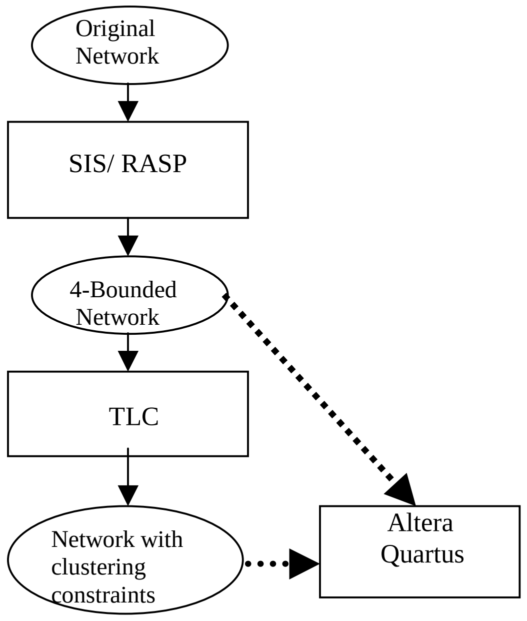

Figure 7 : Experimental flow

In Table 1 we present the results from all four experiments. The device we used was the EP20K600EFC672–1X from the APEX20KE family. According to the APEX device architecture defined in Section 1 we set M_1 to be 10 and M_2 to be 160. The delay model we used was the following: D_1= 0.36 ns, D_2=0.85 ns, D_3 = 1.57ns, *NODE_DELAY*=0.61ns. The data used are extracted from the timing analysis tool used in Quartus. We used combinational circuits from the MCNC and ISCAS benchmark sets.

We see that on the average the maximum delay decreased by 9% when we ran TLC without any node duplications, and by 11% when we allowed partial node duplications. In the latter case the area of the equivalent circuit is increased by 33%. The version with full node duplications provides the best performance results (an average of 15% improvement over the Quartus results) but with a very big area penalty. The results show that our clustering algorithm produces satisfying results compared to a well-known commercial tool. We believe that the improvement is largely due to the use of our two-level clustering formulation and solution. The runtime of our program alone is always under 1 minute for all designs reported in Table 1.

6. CONCLUSIONS & FUTURE WORK

We presented an algorithm for the performance-driven two-level clustering problem, which has application for hierarchical FPGA designs. Our algorithm is performance-driven. We showed that this problem is NP-hard. Experimental results showed that the clustering solution created by our algorithm improved the circuit performance produced by the Quartus System Design from Altera by an average of 15% for APEX devices. To our knowledge this is the first in-depth study for the performance-driven multi-level circuit clustering problem.

Future work can be focused on three areas:

a) Expand the algorithm for sequential circuits. Right now we can only handle combinational circuits.
b) Use a more realistic delay model that considers geometric information. For example the delay between adjacent MegaLABs is sometimes only the half of the delay between distant MegaLABs.
c) Expand the algorithm to handle hierarchical architectures with three or more levels.

7. ACKNOWLEDGEMENTS

This work is partially supported by Altera, Intel, Fujitsu Laboratories of America under the California MICRO program and the NSF Young Investigator Award MIP-9357582. We would like to thank Dr. Chang Wu from Aplus Design Technologies, Inc. for his valuable help.

Table 1: Experimental results (ND stands for node duplication)

Circuit	#LUTs	#I/Os	Quartus	Quartus +TLC (no ND)	%	Quartus +TLC (full ND)	%		%	Quartus +TLC (partial ND)	%		%
			Delay (ns)	Delay (ns)		Delay (ns)		Area (LUTs)		Delay (ns)		Area (LUTs)	
alu	321	22	35.71	33.02	7.5	30.51	14.6	1045	225.6	33.73	5.6	458	42.7
apex2	1152	42	29.49	26.13	11.4	28.22	4.3	1804	56.6	30.82	-4.5	1417	23.0
apex6	390	234	27.00	22.88	15.3	20.93	22.5	571	46.4	21.37	20.9	483	23.8
C1908	146	58	32.32	26.40	18.3	25.41	21.4	827	466.4	27.31	15.5	227	55.5
C5315	537	301	36.44	33.32	8.6	29.43	19.2	2032	278.4	31.21	14.4	776	44.5
C880	166	86	36.30	29.58	18.5	24.71	31.9	364	119.3	27.15	25.2	225	35.5
dalu	430	91	26.01	25.41	2.3	22.62	13.0	811	88.6	26.28	-1.1	496	15.3
des	1569	501	34.86	30.02	13.9	29.65	14.9	3470	121.2	29.87	14.3	2346	49.5
i10	819	481	47.30	42.44	10.3	37.92	19.8	3165	286.5	41.99	11.2	1118	36.5
i9	212	141	25.07	21.32	15.0	21.75	13.3	393	85.4	21.12	15.8	236	11.3
k2	526	88	29.42	29.72	-1.0	25.32	14.0	2258	329.3	27.07	8.0	804	52.9
large	922	41	30.07	30.32	-0.9	26.51	11.8	1654	79.4	26.03	13.4	1194	29.5
misex3	1058	28	26.82	23.70	11.6	21.83	18.6	1903	79.9	22.54	16.0	1318	24.6
too_large	193	41	24.84	24.00	3.4	25.04	-0.8	262	35.8	23.80	4.2	252	30.6
vda	297	56	28.54	24.34	14.7	20.21	29.2	1304	339.1	23.53	17.6	463	55.9
x3	392	234	22.53	22.52	0.0	21.26	5.6	510	30.1	22.29	1.0	438	11.7
Average					9.3		15.8		166.7		11.1		33.9

8. REFERENCES

[1] Brayton R.K., Rudell R., and Sangiovanni-Vincenteli A.L. MIS: A Multiple-Level Logic Optimization, IEEE Transactions on CAD, pages 1062-1081, Nov. 1987

[2] Cong J., Lim S.K., and Wu C. Performance Driven Multi-level and Multiway Partitioning with Retiming, ACM/IEEE 37th Design Automation Conference, Los Angeles, CA, June 2000, pages 274-279.

[3] Cong J. and Xu S. Synthesis Challenges for Next-Generation High-Performance and High-Density PLDs, Asia and South Pacific Design Automation Conf., January 2000, pages 157-162.

[4] Cong J., Li H., and Wu C. Simultaneous Circuit Partitioning/Clustering with Retiming for Performance Optimization in Proc. ACM Design Automation Conf., 1999.

[5] Cong J. and Ding Y. FlowMap: An Optimal Technology Mapping Algorithm for Delay Optimization in Lookup-Table Based FPGA Designs, IEEE Trans. On Computer-Aided Design, 1994, pages 1-12.

[6] Cong J. and Lim S.K. Edge Separability Based Circuit Clustering with Application to Circuit Partitioning. Asia South Pacific Design Automation Conference, Yokohama Japan, Jan.2000, pp.429-434.

[7] Cong J., Peck J., and Ding Y. RASP: A General Logic Synthesis System for SRAM-based FPGAs. ACM/SIGDA International Symposium on Field-Programmable Gate-Arrays, Monterey, California, Feb. 1996.

[8] Cong J. and Romesis M. Performance-Driven Multi-Level Clustering with Application to Hierarchical FPGA Mapping. UCLA CSD Technical Report #010007.

[9] Cook S.A. The complexity of theorem-proving procedures. Proc. 3rd Ann. ACM Symp. On Theory of Computing. New York, 1971, pages 151-158.

[10] Huang D.J. and Khang A.B. When clusters meet partitions: New Density-Based Methods for Circuit Decomposition. In Proc. European Design and Test Conf., pages 60-64, 1995.

[11] Karypis G., Aggarwal R., Kumar V., and Shekhar S. Multilevel Hypergraph Partitioning; Application in VLSI Domain. Proceedings of the 34th annual conference on Design Automation Conference, 1997, pages 526-529.

[12] Lawler E.L., Levitt K.N., and Turner J. Module Clustering to Minimize Delay in Digital Networks, IEEE Transactions on Computers, Vol. C-18, No.1, January 1966, page 47-57.

[13] Murgai R., Brayton R.K., and Sangiovanni – Vincentelli A. On Clustering for Minimum Delay/Area, IEEE International Conference on Computer-Aided Design, November 1991, pages 6-9.

[14] Pan P., Karandikar A.K., and Liu C.L. Optimal Clock Period Clustering for Sequential Circuits with Retiming. IEEE Trans. on Computer-Aided Design, pages 489-498, 1998.

[15] Rajaraman R. and Wong D.F. Optimal Clustering for Delay Minimization, Design Automation Conference, 1993, pages 309-314.

[16] Wei Y.C. and Cheng C.K. Ratio cut partitioning for hierarchical designs. IEEE Trans. on Computer-Aided Design, pages 911-921, 1992.

[17] Yang H.H. and Wong D.F. Circuit Clustering for Delay Minimization under Area and Pin Constraints**,** IEEE Transactions on Computer-Aided Design of Integrated Circuits, September 1997, pages 976-986.