Low-Power Clustering with Minimum Logic Replication for Coarse-grained, Antifuse based FPGAs

Chang Woo Kang and Massoud Pedram
University of Southern California/EE-systems
3740 McClintock ave. EEB-314
Los Angeles, CA 90089, USA
{ckang, pedram}@usc.edu

Abstract

This paper presents a minimum area, low-power driven clustering algorithm for coarse-grained, antifuse-based FPGAs under delay constraints. The algorithm accurately predicts logic replication caused by timing constraint during the low-power driven clustering. This technique reduces size of duplicated logic substantially, resulting in benefits in area, delay, and power dissipation. First, we build power-delay curves at nodes with the aid of the prediction algorithm. Next, we choose the best cluster starting from primary outputs moving backward in the circuit based on these curves. Experimental results show 16% and 20% reduction in dynamic and leakage power dissipation with 18% area reduction compared to the results of clustering without the replication prediction.

Categories and Subject Descriptors

J.6 [Computer-aided engineering]: Computer-aided design (CAD)

General Terms

Algorithm

Keywords

FPGA, Antifuse, Clustering, Power

1 Introduction

FPGAs have become commonplace not only in low-volume designs but also in portable, battery-powered devices craving for power efficiency. These devices with smaller form factor and increased performance continue to define the present and future applications. Applications of this type are characterized as performing faster, becoming smaller in size, having longer battery life, and being marketable ahead of the competition. Previously, programming logic devices were not an option for integration on portable devices because they are bulky and consumed too much power. In the past several years, architectures of FPGAs have improved greatly and finally they play important roles in portable devices.

Antifuse-based FPGAs are one time programmable logic devices. The anti-fuse is initially in a high impedance state and is transformed into a low impedance metal-to-metal link when programmed. Figure 1(b) illustrates the cross-sectional view of the antifuse programming technology. The antifuse element is formed by depositing a high resistance layer (> 1GΩ) of amorphous silicon above a tungsten via a plug that would otherwise bridge the insulation between the two metal layers [1].

Figure 1 shows a coarse-grained, anti-fuse based FPGA from QuickLogic, which is the target device in this paper. The FPGA consists of pASIC3 logic cells, interconnects, and antifuse switches. The logic cell has a large number of inputs and multiple outputs in order to increase the logic utilization. This utilization is however a strong function of the power and efficacy of the design automation tools.

In a typical flow of FPGA CAD tools, clustering, which follows the technology mapping step, is an important optimization because it maps the target circuit net list into an FPGA array. The clustering, therefore, refers to the task of grouping logic gates in the circuit netlist and assigning each group to a configurable logic block in the FPGA array (in the case of our target architecture, this means packing gates into pASIC3 cells.) Logic replication, which is often needed to meet the timing constraints, is an indispensable part of the clustering step. Logic replication directly affects the area and power dissipation of the FPGA synthesis solution. This increase is especially true with respect to leakage power since this leakage is a direct function of the size of the logic circuit implementation. The authors in [8] report that 33% logic replication is observed as a result of the performance-driven clustering in SRAM-based FPGAs.

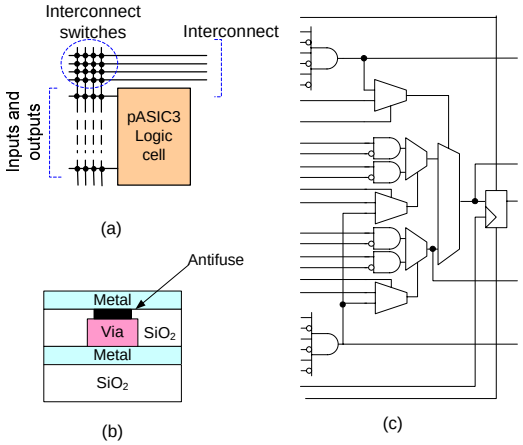

Figure 1: Example of coarse-grained, antifuse-based FPGA: (a) architecture, (b) antifuse switch, and (c) pASIC3 cell.

In this research, we present a low-power driven clustering algorithm with minimal logic replication for coarse-grained, anti-

GLSVLSI'06, April 30–May 2, 2006, Philadelphia, Pennsylvania, USA.

fuse based FPGAs. As stated earlier, in the context of our problem, a *cluster* refers to a group of circuit nodes that can fit in a pASIC3 logic cell. We use a dynamic programming-based clustering technique where starting from the circuit inputs moving toward the circuit outputs, we incrementally generate a set of power-delay curves for all nodes in the network [2][3]. Each such curve, stored at some intermediate node, describes the set of non-inferior clustering solutions for the subgraph rooted at that node. A critical factor in determining the quality of the clustering solution for a circuit is how accurate and complete the set of power-delay curves are; this is strongly depends on the accuracy of incremental cost (i.e., power dissipation) calculation at each node in the network. We have seen that existing heuristics for this cost calculation, which divide the cost of a multiple fanout node equally among its fanout nodes [2][4], can result in significant computational errors, thereby, degrading the quality of the overall solution. In this paper, we present a new heuristic approach for the cost propagation across multiple fanout nodes in a Boolean network that allocates the cost of logic cone rooted at a multiple fanout node to its fanout nodes in proportionately. More precisely, we simply determine the cost allocation to each fanout node it by traversing backward in the circuit, to compute the amount of logic replication.

2 Background

Clustering techniques for SRAM-based FPGAs have been presented in [5][6][7][8], and clustering problem for coarse-grained, anti-fuse based FPGAs has been addressed by a number of researchers [9][10]. In particular, the authors of [10] presented an area-driven clustering algorithm. They set up a pair of linear equations and calculated the minimum number of required pASIC3 logic cells. Their algorithm, which produced 12% area improvement compared to a commercial tool, did not consider the routing cost. The same authors also presented a performance-driven clustering based on a labeling procedure that generates the minimum number of clusters on the timing critical paths. A slack-time relaxation was used to avoid redundant logic replication without violating the performance constraint. In addition, a random merging was used to cluster closely-placed partially-filled clusters. The algorithm gave about 45% delay improvement compared to a commercial tool. The key limitation of their work is that they used a unit delay model, which is not accurate enough to estimate the delay of a logic design. A delay-optimal clustering for low power was presented in [3]. For optimality, they enumerate all feasible cluster patterns at each gate in the circuit and maintain only the power-optimal solutions at each gate for each arrival time value.

In this research, we follow the same flow as that in [10]. There are four different programmable gate groups (cf. Figure 2) inside a *pASIC3* logic cell. We call each of these gate groups a *base gate*. After deriving the base gates, cell generation is performed for each base gate. Cell personalization is done either by assigning constant 1 or 0 to some of the inputs or by connecting (bridging) some of the inputs together. By applying all possible combinations of these two operations to a base gate, many different library cells can be generated. We call the personalized cells "*primitive cells*".

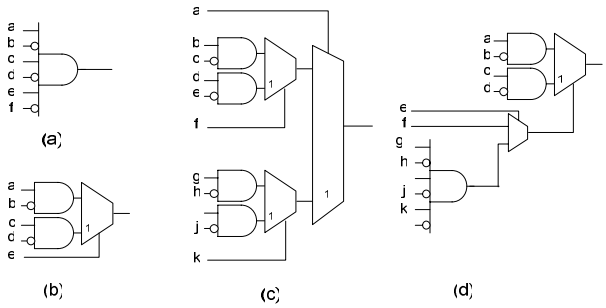

Figure 2: Base gates extracted from pASIC3 cell: (a) base-gate A, (b) base-gate B, (c) base-gate C, and (d) base-gate D.

3 Design Flow and Problem Description

A *cluster i*, denoted by CL_i, is defined as a group of circuit nodes that can be realized in a single pASIC3 logic cell without any resource conflicts. The set of nodes that drive nodes in cluster CL_i is referred to as its **leaf set** and denoted by Λ_i.

The clustering algorithm comprises of two steps: cluster generation and cluster selection. During the cluster generation, clusters rooted at nodes in network are generated and power-delay curves are computed in a postorder traversal of the network starting from primary inputs going toward the primary outputs. For cluster selection, clusters are determined during a preorder traversal from primary outputs back toward the primary inputs. The design flow can be described as follows:

1. Select the logic cone rooted at a primary output, which has the largest number of un-clustered nodes.
2. Traverse the cone in postorder to create power-delay (PD) curves.
3. Select a power-delay point from the PD curve of a primary output and form a cluster based on the point.
4. Select power-delay points from PD curves at leaf nodes of the previous cluster and do this in preorder until all nodes in the logic cone are clustered.
5. Go to step 1 if any logic cone is not clustered yet.

A clustering solution at a node u is characterized by a power-delay point (PD-point) which is a pair $\{p_u, d_u\}$, where d_u gives the delay value (i.e., latest signal arrival time) associated with the PD-point, and p_u gives the corresponding power dissipation of the clustering solution rooted at node u.

Consider intermediate nodes n_i and n_j in a Boolean network (a circuit netlist with signal direction specified) where there exists a common multiple fanout node, n_k, in their transitive fanin cones. In typical performance-driven clustering, to minimize the arrival time to n_i and/or n_j, logic replication of logic under n_k may become necessary. An example of this scenario is shown in Figure 3(a) where when finding clustering solutions at nodes n_4 or n_5, it may become necessary to replicate n_1, n_2 and n_3. Assume that there are two possible clustering solutions, CL_2 and CL_3 (CL_4 and CL_5), at n_4 (n_5). [1] There is also a single clustering solution CL_1 at node n_3. The area of each cluster is 1 whereas the delay depends on the topology of the logic mapped to the cluster. We calculate the AD curve of n_4 as follows. For the clustering solution CL_3, the AD value is (1,0.8) whereas for CL_2, the area value is 1+1/2=1.5 and the delay is 1. The area cost calculation is done in this way because the cost of cluster CL_1 is divided equally between its two

[1] In this example, because the area cost is easier to depict pictorially, the area cost is used in place of the power cost.

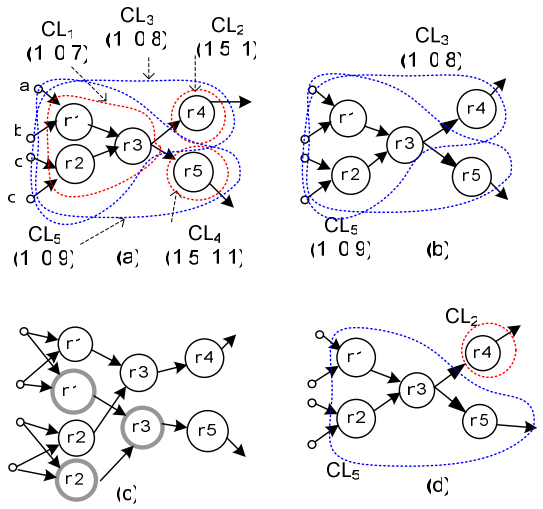

Figure 3: An example of redundant logic replication in clustering: (a) clusters and the corresponding area-delay points, (b) non-inferior clusters, (c) circuit after logic replication (i.e., n_1, n_2, and n_3 are duplicated), and (d) a desired clustering solution.

fanout nodes. This generates a new AD value of (1.5,1). Notice however that (1.5,1) is inferior to (1,0.8), and therefore, it will be dropped, resulting in the AD curve of {(1,0.8)} for n_4. [2] Similarly, the AD curve of n_5 will be pruned to {(1,0.9)}. However, by dropping the two inferior points from the AD curves of n_4 and n_5 we force a logic clustering solution whereby three nodes (n_1, n_2 and n_3) must be replicated as shown in Figure 3(b) and (c). The overall area cost of this clustering solution is 2 and the worst-case delay is 0.9. Suppose that the required time at node n_4 and n5 is 2. Now, in fact, there is a better solution whereby CL_2 is chosen at n_4 and CL_5 at n_5 (cf. Figure 3(d).) The area cost of this solution is 2, while its worst-case delay cost is 1.1. However, there is no logic duplication, which means that the utilization of one of the pASIC3 logic cells in the latter solution is much lower, thereby, potentially allowing a future packing of extra logic into that pASIC3 cell. The reason that the area cost of the solution given in part (d) is 2 is that CL_5 can be treated as multiple-output Boolean functions providing both the signal that goes out of n5 and the signal that goes out of n_3 and feeds into cluster CL_2. Therefore, there is no need to replicate n_1, n_2 and n_3 to separately generate the signals from n_3 into CL_2, as would have been the case if the cluster was treated as a single-output Boolean function.

We have identified the aforesaid problem as a key reason behind a significant increase in the logic replication cost of a mapping solution to pASIC3 arrays. Therefore, in the remainder of this paper, we focus on developing a heuristic solution to calculate the replication cost across multiple-fanout nodes of the circuit during the post-order traversal.

4 Performance-driven Clustering

In this section, we present a clustering procedure with the accurate calculation of logic replication cost during the forward traversal of the Boolean netlist.

[2] An inferior point (p', d') is inferior to (p, d) if ($p' \geq p$ and $d' > d$) or ($p' > p$ and $d' \geq d$).

4.1 Cluster generation and power-delay curves

A technology mapped network consists of primitive cells. In the cluster generation phase, we postorder from the primary inputs to the primary outputs. This ordering ensures that when a node is processed, all of its fanin nodes have already been processed. When constructing the PD curves for some node, n, we first invoke a *matching algorithm* described in [12] to enumerate all possible cluster matches at that node. For each cluster match, we then calculate its PD value as follows. The (dynamic programming) power value of the cluster is the summation of the (dynamic programming) power values of all its inputs plus the power cost of the cluster itself. Similarly, the (dynamic programming) delay value of the cluster match is the maximum of the (dynamic programming) delay values of its inputs plus the delay thru the cluster itself.

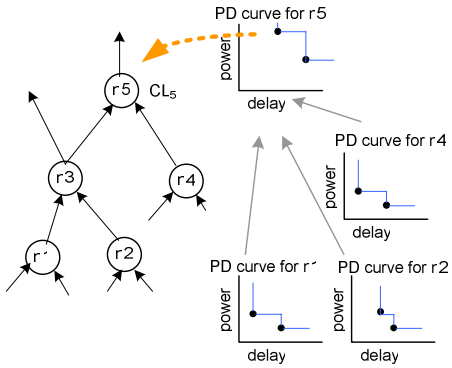

Figure 4: PD curve generation for a node with a cluster.

Figure 4 illustrates the PD curve generation at node n_5 with a cluster CL_n. PD curves of leaf set nodes n_1, n_2, and n_4 have already been computed. The PD curve for CL_n matched at node n_5 is created by PD curves from the leaf nodes. In the conventional calculation method of [2][4], the power dissipation at node n for cluster match CL_n is calculated as:

$$P_{dyn}(n, CL_n) = \frac{1}{2} V_{dd}^2 \times f \times \sum_{u \in nodes\ in\ CL_n} C_{fo}(u) sw_u + \sum_{n_i \in inputs(n)} \left(\frac{P_{dyn}(n_i, CL_{n_i})}{fanout(n_i)} \right) \quad (1)$$

where $C_{fo}(u)$ is the capacitance driven by node u, sw_u is the transition probability of node u, and $fanout(n_i)$ is the number of fanouts that node n_i drives. The arrival time at node n_5 with CL_n is simply the maximum arrival time among arrival times from leaf nodes plus the delay thru the cluster.[3]

4.2 Correct accounting of logic replication

Logic replication may be needed to meet a timing constraint at a node. It occurs when a selected cluster rooted at the node covers nodes that have already been covered by another cluster. Logic

[3] For pASIC3 mapping problem, there is no "unknown load problem" [4], which often complicates the calculation of the dynamic programming power and delay values in ASIC design flows. This is because the load ahead of a node during the post-order traversal is always the load imposed by another pASIC3 logic cell. Notice that the input pin capacitances of all inputs to a pASIC3 logic cell are the same.

replication potentially occurs on the boundary of logic cones associated with primary outputs.

We propose an algorithm, which estimates the cost of logic duplication by simulating the clustering procedure for each PD point during the postorder traversal. The algorithm assumes that the delay of each PD point at a node is close enough to the required time at the node. Notice that, given the required time at a node, the best PD point has the largest delay, which is equal to or less than to the required time, and the smallest cost. Therefore, being selected as the best PD point means that the required time at that node is very close to the delay in the PD point. Therefore, we use the delay in the PD point as the required time at the node. Under this assumption, being aware that the maximum path delay from a fanin node in a cluster are the largest delay from the fanin node to the root node of the cluster, we can calculate required times of fanin nodes of a cluster by subtracting the maximum path delays from the required time at the root node. If any required time of fanin nodes, which has been covered by clusters, is equal to or larger than the arrival time of the fanin, there is no logic duplication on the logic cone boundary. This leads to zero cost toward transitive fanin of the crossing boundary, whereas typical approach divides the cost by the size of fanout of fanin nodes. If logic duplication is mandatory to meet the timing constraint, we only add the cost caused by the duplicated logic. Notice that duplication operation can go toward primary inputs until no timing violation occurs. Let's assume that logic cone PO0 has been covered by clusters, and node d has a PD point having a node b and e as fanin nodes. Figure 5(a) depicts the case in which no duplication is necessary. Since our approach does not account for the cost of unduplicated nodes, the cost toward transitive fanin of node b is zero. Therefore, we simply add the cost at node e and the cost of node d to the total cost at node d. On the other hand, if duplication is required as shown in Figure 5(b), the cost caused by the duplicated nodes is added to the total cost.

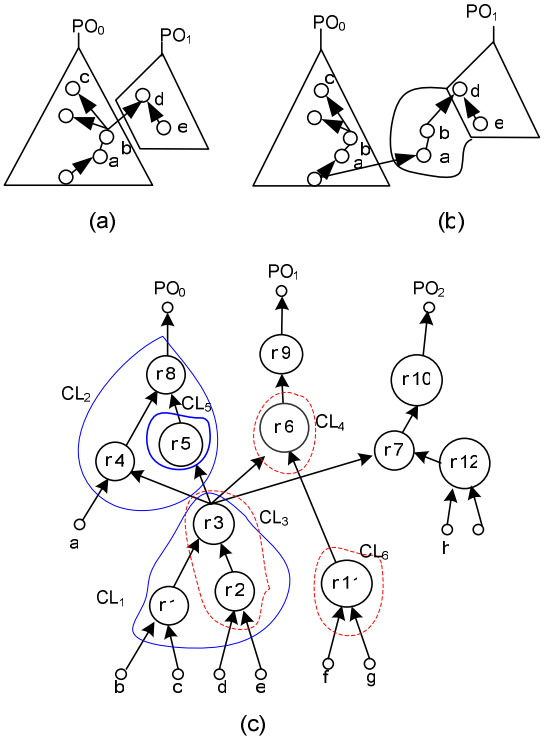

Figure 5: Example of logic replication prediction.

An example in Figure 5(c) illustrates this notion in detail. An un-clustered logic cone, $\Phi(PO_i)$, is defined as the set of un-clustered nodes in the transitive fanin cone of primary output, PO_i. For the moment, only focus on the solid closed curves and ignore the dashed ones. In Figure 5(c), $\Phi(PO_0)$ is clustered first. In our proposed heuristic accounting of the logic replication cost during the postorder traversal of the circuit graph, when calculating the dynamic programming (DP) power cost of CL_5 at n_5, we divide the DP cost of CL_1 at n_3 by its fanout count inside the logic cone (which is two) and add to this quantity the power cost of CL_5. Note that when we calculate the DP cost of CL_2 at n_8, we would account for the cost of CL_1 exactly once (1/2 contribution coming from the $n_3 \rightarrow n_5$ branch, the other coming from the $n_3 \rightarrow n_4$ branch.)[4] Suppose that after preorder traversal of logic cone $\Phi(PO_0)$, we select a clustering solution in which CL_1 is matched at n_3 while CL_2 is matched at n_8. Next, we start clustering logic cone $\Phi(PO_1)$. Consider generating the PD curve at node n_6 (having first processed node n_{11}, creating a cluster match of CL_6 at that node.) For cluster CL_4 matched at node n_6, we need to compute the DP power cost of its fanin nodes n_3 and n_{11}. At n_3 (n_{11}), we have the PD curve of all possible clustering solution rooted there. However, we do not know what specific clustering solution for the cone rooted at n_3 will be used for each PD point at n_6. This is the key difficulty in the estimation of logic replication cost. Consider two extreme cases where in one case, CL_1 match at n_3 is used as the best solution for $\Phi(PO_1)$ resulting in no logic duplication; in the other case all of the cone under n_3 is replicated since no common signals exist between the best matching solution of this sub cone under $\Phi(PO_0)$ and $\Phi(PO_1)$. The way we solve this problem is to calculate the PD curve of CL_4 matched at node n_6, by completely ignoring the fact that cone $\Phi(PO_0)$ has already been processed and a mapping solution has been obtained. Suppose a PD curve of $X=\{x1,x2,\ldots,xm\}$ at node n6 is generated in this way, where $xi=(pi,di)$. Take any point say xi corresponding to a clustering solution with CL4 matching at n_6. We assume that di is the required time at n_6. We go ahead and calculate the required time at output of n_3 as di-delay(CL_3). If this required time is larger than the arrival time at n_3 coming from the synthesis solution for $\Phi(PO_0)$, then for the calculation of the DP power cost of cluster CL4 at n_6, the DP power cost of subcone rooted at n_3 is set to zero. Otherwise (i.e., a timing violation will occur if the solution generated for $\Phi(PO_0)$ is used), we find the optimum clustering solution of logic subcone rooted at n_3 and use the power cost of this solution toward the calculation of the power cost of cluster CL_4 at n_6. The dashed enclosed curves show a case in which the subcone rooted at n_3 must be resynthesized in order to meet a timing requirement at PO_1. Notice that in case of logic duplication, the duplicated copy of n_3 needs to drive only node n_6; therefore, the PD curve at node n_3 must be updated to reflect this change in load. The arrival time of CL_4 becomes the maximum value among arrival times of different input paths. Arrival times through duplicated nodes can be calculated based on arrival times of clustered nodes.

[4] By reducing the power dissipation contribution of CL_1, we tend to favor an overall clustering solution in which multiple fanout nodes are preserved after mapping, which reduces logic replication and improves the final mapped power dissipation as was done in [2].

Algorithm predict_logic_duplication(*node, Network*)

1. *PD_curve* = read_pd_curve(node)
2. **for** each point *p* for *PD_curve*
3. Λ = leaf_set_of(*p*)
4. **if** any node in Λ is not clustered
5. continue
6. **end if**
7. *r = p.delay*
8. compute_load_cluster(*p.cluster*)
9. compute_required_time(Λ, *r*)
10. **for** each node *u* for Λ
11. *cycle_time = u.required_time*
12. predict_cluster_selection(*u, p.cluster, cycle_time*)
13. **end for**
14. **end for**

Algorithm predict_cluster_selection(*node, ParentCluster, cycle_time*)

1. if *node* is a primary input or *node* is not clustered
2. return
3. **end if**
4. update_arrival_time(*node*)
5. **if** *node.arrival* $\leq$ *cycle_time*
6. return
7. **end if**
8. update_pd_curve(*node*)
9. *p* = get_best_point(*node.PD_curve*)
10. Λ = leaf_set_of(*p*)
11. compute_load_cluster(*p.cluster*)
12. compute_required_time(Λ, *cycle_time*)
13. **for** each node *u* for Λ
14. *cycle_time = u.required_time*
15. predict_cluster_selection(*u, p.cluster, cycle_time*)
16. **end for**

Figure 6: Prediction of logic replication.

Accounting for logic replication, the total power dissipation at node n with cluster CL_n can be extended from equation (1) and can be given by:

$$P_{dyn}(n, CL_n, \Phi_n) = \frac{1}{2} V_{dd}^2 \times f \sum_{u \in nodes\ in\ CL_n} C_{fo}(u)\, sw_u + \sum_{n_i \in inputs(n)} \left(\frac{P_{dyn}(n_i, CL_{n_i}, \Phi_n)}{fanout(n_i, \Phi_n)} \right) \quad (2)$$

where Φ_n is a logic cone to which node n belongs, $C_{fo}(u)$ is the capacitance driven by node *u*, sw_u is the transition probability of node *u*, and *fanout(n,* Φ_n*)* is the number of fanouts inside Φ_n from node n.

Figure 6 gives the pseudo code to account for the effect of logic replication. When a node has to select a cluster, the function predict_logic_replication is executed. It first checks to see if the replication is necessary by checking if any node in leaf set has been clustered. In order to compute the required times for nodes in the leaf set, the capacitance of a node is computed as if the node has been clustered. The required times for nodes in leaf set are computed and passed on to the next level logic replication prediction in the recursive function predict_cluster_selection.

5 Cluster selection

After the PD curves for all nodes in the transitive fanin cone of a primary output are computed during the postorder traversal of the circuit, a suitable point on the PD curve of the root node is chosen, given the required time at the root of the logic cone. The cluster for the point at the root is identified and the required times for its inputs are computed. The preorder traversal resumes at its child nodes to satisfy the new required time while minimizing the power dissipation. Our approach is similar to PDMAP presented in [2].

6 Implementation and Experimental Results

We have implemented the clustering algorithm based on SIS [11] and used 90nm CMOS technology process model to estimate delay and power dissipation information of primitive cells. Large combinational circuits were selected from the MCNC91 benchmark. We first ran low power technology mapping by using PDMAP [2] and then applied our low power, minimal logic replication clustering algorithm to the network.

In FPGAs, inter-cluster interconnect capacitance interconnect is not ignorable. Thus, we use constant values representing those capacitances in pASIC3 family FPGAs. However, in this research we assumed that the capacitances of intra-cluster interconnect is ignorable. The key limitation of the present work is that of assuming a fixed capacitance of inter-cluster interconnections. Inter-cluster interconnect is a major component in FPGAs and accurately estimating the capacitance is crucial in early stages of the design in order to increase the efficacy of this clustering algorithm.

Table I shows the experimental results. Our approach could reduce the total number of nodes by 18% on average, resulting in savings in area and power dissipation without any sacrifice of speed. The run time increases due to the repeated invocation of the logic replication predictor for the same node with the same required.

7 Conclusion

In this paper, a minimal-area clustering algorithm for low power was proposed. The proposed algorithm builds PD curves for nodes in a network by predicting the amount of logic replication based on the timing constraint. The prediction provides accurate power dissipation on a cost point on the curves.

Experimental results indicate that the proposed algorithm generates much less duplicated logics with less delay and power dissipation compared to the traditional cost distribution method. The algorithm achieved 18% reduction on the total number of nodes, resulting in saving both dynamic power and leakage power dissipation by 16% and 20% respectively without any sacrifice of delay.

References

[1] pASIC3 FPGA Family Datasheet, QuickLogic Corporations (http://www.quicklogic.com).

[2] C. Tsui, M. Pedram, and A. M. Despain, "Technology decomposition and mapping targeting low power dissipation," in Proc. Design Automation Conference, 1993, pp. 68-73.

[3] H. Vaishnav and M. Pedram, "Delay-optimal clustering targeting low-power VLSI circuits," IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 18, no. 6, pp. 799 – 811, 1999.

[4] K. Chaudhary and M. Pedram, "Computing the area versus delay trade-off curves in technology mapping," IEEE Trans. on Computer Aided Design, Vol. 14, No. 12, 1995, pp. 1480-1489.

[5] J. Cong, J. Peck, and Y. Ding, "RASP: a general logic synthesis system for SRAM-based FPGAs," in Proc. FPGA, pp. 137 – 143, 1996.

[6] V. Betz and J. Rose, "Cluster-based logic blocks for FPGAs: area-efficiency vs. input sharing and size," in Proc Custom Integrated Circuits Conference, 1997, pp. 551 – 554.

[7] Alexander Marquardt, Vaughn Betz, and Jonathan Rose, "Using cluster-based logic blocks and timing-driven packing to improve FPGA speed and density," in Proc. FPGA, pp. 37-46, 1999.

[8] J. Cong and M. Romesis, "Performance-driven multi-level clustering with application to hierarchical FPGA mapping," in Proc. Design Automation Conference, 2001, pp. 389 - 394.

[9] C-W. Kang, A. Iranli, and M. Pedram, "Technology mapping and packing for coarse-grained, antifuse-based FPGAs," in *Proc. Asia and South Pacific Design Automation Conference*, 2004, pp. 209 - 211.

[10] C-W. Kang and M. Pedram, "Clustering techniques for coarse-grained, antifuse-based FPGAs," in *Proc. Asia and South Pacific Design Automation Conference*, 2005, pp. 785 - 790.

[11] E.M. Sentovich, et al., SIS: A system for sequential circuit synthesis, 1992, Electronics Research Laboratory, College of Engineering, University of California, Berkeley.

[12] C. M. Hoffman and M. J. O'Donnell, "Pattern matching in trees," *Journal of the Association for Computing Machinery*, pp. 68 - 95, 1982.

Table I: Low-power clustering results

	Without replication prediction						With replication prediction					
Ckts	Nodes	Clusters	Delay (ns)	Dynamic power (μW)	Leakage power (μW)	CPU time (s)	Nodes	Clusters	Delay (ns)	Dynamic power (μW)	Leakage power (μW)	CPU time (s)
i9	432	188	0.57	317	362	11	440	195	0.55	314	284	17
rot	514	189	0.46	380	450	6	392	175	0.43	306	336	14
i8	721	336	0.62	492	610	18	620	319	0.62	425	521	27
pair	1135	422	0.64	809	996	28	914	398	0.58	686	791	91
vda	563	206	0.36	217	514	14	408	161	0.34	149	368	16
x1	229	87	0.14	179	195	1	170	75	0.15	136	142	1
C5315	1016	391	0.47	866	872	9	883	378	0.46	748	757	18
alu4	672	224	0.68	354	594	36	545	224	0.68	293	471	116
apex6	546	170	0.27	346	477	4	419	174	0.29	295	355	6
C880	317	100	0.66	226	280	6	262	112	0.67	201	230	26
C3540	1085	357	0.88	692	954	49	858	346	0.85	577	735	205
alu2	343	112	0.48	212	295	13	276	108	0.47	172	234	10
C1355	306	109	0.33	229	266	3	271	96	0.32	174	237	6
C1908	301	108	0.50	203	263	3	241	87	0.49	159	210	7
C499	306	109	0.33	229	266	2	271	96	0.32	174	237	8
	1	1	1	1	1	1	0.82	0.95	0.98	0.84	0.80	2.8