Clustering Techniques for Coarse-grained, Antifuse-based FPGAs

Chang-woo Kang and Massoud Pedram

Abstract

In this paper, we present area and performance-driven clustering techniques for coarse-grained, antifuse-based FPGAs. A macro logic cell in this class of FPGAs has multiple inputs and multiple outputs. Starting with this macro cell, a library of small logic cells can be generated and a target network was mapped with the library. For the minimum-area clustering, our algorithm minimizes the number of required macro logic cells to cover a network. Two linear equations were set up and we found the optimal mapping solution by using the equations. For the performance-driven clustering, the number of macro logic cells on the critical path is minimized by using the extension of Lawler's algorithm. The results show that the area-driven clustering algorithm reduced the number of macro logic cells by 12.29% and the performance-driven clustering reduced the maximum depth by 44.75% compared to a commercial tool.

1. Introduction

Field programmable gate arrays (FPGAs) can provide many advantages over standard cells. Fast time-to-market satisfies industry designers to keep up with newly created standards, and configurability provides flexible hardware on demand of both new standards without fabricating a new chip. On the other hand, there are some aspects, which must be significantly improved in the near future. Area, speed, and power dissipation are still far behind standard cells. The ratios of area, speed, and power dissipation of SRAM-based FPGAs compared to static CMOS implementation are 10x, 3x, and 100x, respectively, according to [7] and [8].

Coarse-grained, antifuse-based FPGAs have emerged as a promising technology for small area, high speed, and low power. Figure 1 shows a coarse-grained, antifuse-based pASIC3 logic cell [9], which has 29 inputs including the clock and five outputs. The function of the logic cell is determined by the logic levels applied to the inputs of the AND gates and multiplexers. The high logic capacity and fan-in of the logic cell accommodate many user functions with a single level of logic delay. Coarse-grained, antifuse-based FPGA architecture demands highly intelligent CAD algorithms, because the architecture provides tremendous flexibility with small hardware overhead. For example, the size of an antifuse, to connect two metal wires is smaller than *a via* [9].

In this paper, we present both area-driven and performance-driven clustering techniques. Even we target specific logic cell architecture, our method can be applied to similar type of coarse-grained, antifuse FPGAs with slight modification. We have divided it into several base gates and then mapped a network. After technology mapping, we found the minimum number of macro logic cells to cover the network by setting up two linear equations. From the equations, we found either the minimum crossing points or the minimum value under certain ranges. For performance-driven clustering, we minimized the number of pASIC3 logic cells on the critical path by optimal labeling and clustering. Slack-time relaxation was applied to minimize logic duplication without violating the maximum required labels at primary outputs. Merging was done by randomly selecting a cluster and greedily merging closely located clusters.

This paper is organized as follows: In section 2, a brief background is provided. The area-driven clustering algorithm is presented in section 3. The performance-driven clustering algorithm is discussed in section 4. In section 5, experimental results are provided. Finally, we conclude in section 6.

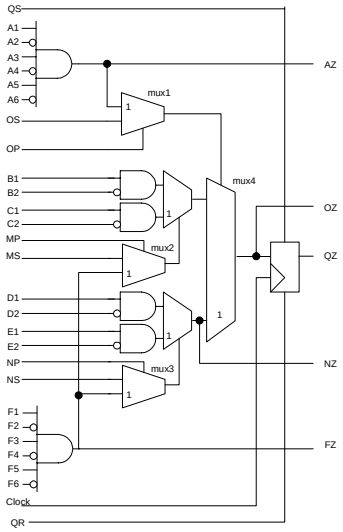

Figure 1: pASIC3 logic cell

2. Background

Clustering techniques for two different technologies, SRAM and antifuse, are somewhat different. A lookup table is a universal function table, which can realize any function if and only if the number of inputs of the function is not larger than that of the lookup table. The number of lookup tables inside a macro programmable block limits clustering. However, a coarse-grained, antifuse macro logic cell consists of gates connected by multiplexers and the logic cell is not universal, as shown in Figure 1. Since it is too difficult to map a network with multiple output logic cells, the macro logic cell must be divided into small base gates and library cells are generated from those base gates. After technology mapping, the library cells must be packed to fit the macro logic cell. The constraint for packing is more stringent.

Clustering techniques for SRAM-based FPGAs have been evolving [0]-[13]. The algorithms relied on good seed selection and smart gain functions to evaluate gain of absorbing a neighbor node according to their objectives. A dynamic programming method was presented to find the minimum number of pAISC3 logic cells to cover a mapped network [14]. The pASIC3 logic cell was divided into several base gates, and library cells were generated from those base gates by bridging inputs or sticking inputs to V_{dd} or GND. Each cell was generated from different base gates and depending on which base-gates generated the cell, the cell type was determined. Since there can be different combinations to compose a pASIC3 logic cell, all combinations could be enumerated. An extension of coin-change problem, which is solvable by a dynamic programming, gave the optimal solution.

For this research, we have generated the library cells as in [14]. Therefore, we adopt the terminology of that reference. There are four different programmable gate groups inside a *pASIC3* logic cell.

We call each of these gate groups a *base gate* as shown in Figure 2. After deriving 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 some of the inputs together. We call the former operation "*sticking*" and the latter operation "*bridging*". By applying all possible combinations of these two operations to a base gate, many different library cells can be generated. We call those personalized cells "*primitive cells*". However, some of the primitive cells generated from different base gates may have the same Boolean function. In fact, we can draw a Venn's diagram to depict the set relationship among the primitive cells that are generated from different base gates, as depicted in Figure 3. Note that the total number of primitive cells of any type is more than 5,000. Finally, we filtered out rarely-used primitive cells based on experiments with MCNC91 benchmark circuits. Consequently, we selected 886 primitive cells shown in Table 1. All library generation processing has been automated with perl scripts.

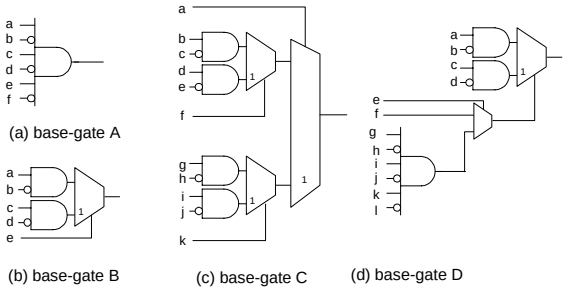

Figure 2: Base gates extracted from pASIC3 logic cell

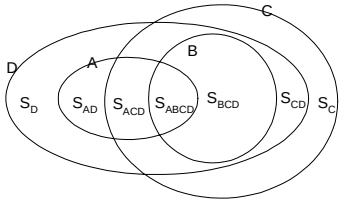

Figure 3: Venn's diagram of for the set of cells that can be personalized from base gates

Table 1: Cell distribution for the selected primitive cells.

Type set	S_{AD}	S_{ACD}	S_{BCD}	S_C	S_D	S_{CD}	S_{ABCD}
Cells	3	5	20	714	110	28	6

3. Area-Driven Clustering

In this section, we provide an algorithm to find the minimum number of pASIC3 logic cells to cover a mapped network.

3.1 Problem Statement

After a mapped netlist is generated after technology mapping, we must solve the problem of clustering the primitive cells used in the mapped netlist to the pASIC3 logic cells. Routing cost of the connections in the mapped netlist tends to be large for the FPGAs. Since the mapping is performed before placement and routing, physical information is not available. In addition, antifuse-based FPGAs have relatively rich routing resources since routing switches are abundant and many layers of metal wires can cross over the pASIC3 logic cells [9]. Thus, we opted to minimize the total area taken by the pASIC3 logic cells during the clustering and pASIC3 assignment step.

Problem 1: Given a mapped netlist, we want to find the minimum number of *pASIC3* logic cells that can realize the network.

3.2 Set containment relations

Base gates can be put into two classes: *simple and complex base gates*. The complex base gate is one that consists of multiple base gates and internal multiplexers, while the simple base gate cannot be composed by other base gates. Base-gates C and D are complex, whereas base-gates A and B are simple. The inclusion relationship between these base-gates is expressed as follows:

$$\begin{aligned} &basegate\ B \subset basegate\ C \\ &basegate\ B \subset basegate\ D \\ &basegate\ A \subset basegate\ D \end{aligned} \qquad (1)$$

Notice that when both a simple base gate and a complex base gate can implement a primitive cell, the simple base gate will be selected for realizing the function of the primitive cell. Realizing the function by the complex base gate not only wastes area of the pASIC3 logic cell but also needlessly increases the circuit delay. Therefore, we can safely state that base-gates C and D are inferior to base-gates A and B when they implement the same logic function.

Internal multiplexers in *pASIC3* logic cells were used to divide the logic cell into four base gates. Sometimes, by utilizing these internal multiplexers, a collection of primitive cells may be transformed to a single primitive cell of a different type. The transformation will reduce area, delay, and power dissipation by utilizing internally hard-wired connections inside the pASIC3 logic cell.

3.3 Minimum number of pASIC3 logic cells with given base gates

Given the number of base gates for each type, the key question is how many pASIC3 logic cells are required to contain all of the base gates. There are three types of pASIC3: *2A + 2B*, *2A+C*, and *A+B+D*. A type 2A+2B pASIC3 logic cell is defined as the pASIC3 logic cell that has two base-gate A's and two base-gate B's in it. Other types can be defined similarly. Note that only one base-gate D can fit in one pASIC3 logic cell from its logic architecture in Figure 1.

Theorem 1: Let n_A denote the number of base-gates A. n_B, n_C, and n_D are similarly defined. The minimum number of pASIC3 logic cells N_{pASIC3} needed to implement a mapped netlist containing n_A, n_B, n_C, and n_D base-gates can analytically be calculated as follows:

$$\begin{aligned} N_{pASIC3} &= \max\left(N_{p1}, N_{p2}\right) \\ N_{p1} &= \begin{cases} \dfrac{n_A + n_D}{2} & ,if\ n_D < n_A \\ n_D, & otherwise \end{cases} \\ N_{p2} &= \begin{cases} \dfrac{n_B + n_D}{2} + n_C, & if\ n_D < n_B \\ n_D + n_C, & otherwise \end{cases} \end{aligned} \qquad (2)$$

Proof: At most two base-gate A's can be packed inside a single pASIC3 logic cell. Similarly at most one base-gate D can be packed inside a single logic cell but it uses only one base-gate A. Therefore, if one base-gate D is packed, there exists an empty space for an base-gate A. When n_A is larger than n_D. The lower bound on the number of pASIC3 logic cell is $N_{p1} = \frac{n_A - n_D}{2} + n_D = \frac{n_A + n_D}{2}$. However, when n_A is equal to or less than n_D, since there should be enough spaces for base-gate A's, which are not occupied by base-gate D's, the low bound becomes $N_{p1} = n_D$. Similar argument yields

another lower bound: $N_{p2} = \frac{n_B - n_D}{2} + n_C + n_D = \frac{n_B + n_D}{2} + n_C$ for $n_D < n_B$ and $N_{p2} = n_D + n_C$ for $n_D \geq n_B$. Clearly, the overall lower bound is the maximum of these two bounds, which is the desired result. ■

3.4 Type distribution table

Theorem 1 can be used to significantly simplify the problem. After technology mapping, we count the number of primitive cells of specific types. Let ns_Γ denote the number of the primitive cells of type Γ in the mapped network. For example, ns_{AD} is the number of type-AD primitive cells, i.e., the number of those cells that belong to set S_{AD}. The problem can be restated follows:

Problem 2: Given a primitive cell library generated from the pASIC3 logic cell structure and a mapped network comprising of the primitive cells, we want to find the best choices of base gates A, B, C and D for realizing all of the primitive cells in the network so as to minimize the number of required pASIC3 logic cells.

Note that after the base gate counts are known, the minimum number of logic cells can be computed straightforwardly based on Theorem 1.

Table 2: The type distribution table for primitive cell to base-gate mapping.

# of primitive cell types	# of Base-gate types			
	A	B	C	D
ns_{AD}	ns_{AD}	0	0	0
ns_{ACD}	x	0	$ns_{ACD} - x$	0
ns_{BCD}	0	ns_{BCD}	0	0
ns_D	0	0	0	ns_D
ns_C	0	0	ns_C	0
ns_{CD}	0	0	$ns_{CD} - y$	y
ns_{ABCD}	z	$ns_{ABCD} - z$	0	0

Table 2 shows how a primitive cell of type Γ in the mapped network is realized with a base gate of type A, B, C, or D. Notice that many of the primitive cell types have a unique realization in a single base-gate type. Examples include types BCD of primitive cells. Note that a type BCD primitive cell should be realized only using type B base gates because of the inclusion relationship of equation (1) and the fact that complex base-gates are always more costly than the corresponding simple base gates. Three of the primitive cell types, however, can be realized by using either of two base gates. For example type ACD primitive cell can be realized as either type A or type C base gate. This table shows that, to solve problem 2, all we have to do is to determine variables *x*, *y* and *z* where *x* denotes the number of primitive cells of type ACD that are realized as a type A primitive gate, y denotes the number of primitive cells of type CD that are realized as a type D primitive gate, and z denotes the number of primitive cells of type ABCD that are realized as a type A primitive gate.

Problem 3: Given the occurrence count of different primitive cell types in a mapped network, find the values of variables x, y and z so as to minimize the number of pASIC3 logic cells required to cover the network.

3.5 Problem formulation and solution

We formulate Problem 3 as a linear programming problem and then obtain the optimal solution by finding either the minimum point of an intersected plane of two equations [0] or the minimum point of an equation that is always above the other within certain ranges of variables. Equation (2) can be restated as in (3).

$$
\begin{aligned}
&N_{pASIC3} = \min\left\{\max\left(N_{p1}(x,y,z), N_{p2}(x,y,z)\right)\right\} \\
&0 \le x \le ns_{ACD}; 0 \le y \le ns_{CD}; 0 \le z \le ns_{ABCD} \\
&N_{p1} = \begin{cases} \frac{1}{2}\left(ns_{AD} + x + z + ns_D + y\right), & if\ ns_D + y < ns_{AD} + x + z \\ ns_D + y, & otherwise \end{cases} \\
&N_{p2} = \begin{cases} \frac{1}{2}\left(ns_{BCD} + ns_{ABCD} - z + ns_D - y\right) + \left(ns_{ACD} - x + ns_C + ns_{CD}\right), & \\ \qquad\qquad if\ ns_D + y < ns_{BCD} + ns_{ABCD} - z & \\ ns_D + ns_{ACD} - x + ns_C + ns_{CD}, otherwise & \end{cases}
\end{aligned}
\tag{3}
$$

The brute-force algorithm is to search for the optimal solution by trying out every possible combinations of x, y, and z within their allowed ranges ($0 \le x \le ns_{ACD}$, $0 \le y \le ns_{CD}$, $0 \le z \le ns_{ABCD}$). The computational complexity, however, is $O\left(ns_{ACD} \times ns_{CD} \times ns_{ABCD}\right)$, which can be quite high. Fortunately, equation (3) has an important property that allows us to speed up the search: As *x*, *y*, and *z* increase, N_{p1} increases but N_{p2} decreases. Therefore, within allowed ranges of *x, y, and z*, equations for N_{p1} and N_{p2} may intersect in a plane or one equation is above the other all the time. We explain the solution for the two cases as follows.

Case 1: When N_{p1} and N_{p2} intersect in a plane, at the intersected plane, N_{p1} and N_{p2} become equal:

$$F(x,y,z) = N_{p1}(x,y,z) - N_{p2}(x,y,z) = ax + by + cz + d = 0 \tag{4}$$

where *a*, *b*, *c*, and *d* are coefficients of an equation of a plane after the subtraction. All points in this plane guarantee that logic cells are full because N_{p1} and N_{p2} are equal but choosing one arbitrary point on the plane may not give the optimal solution. Therefore, we need to find the point that gives the optimal solution in this plane. Notice that we should consider only points on the plane within the specified ranges for *x*, *y*, and *z*. Further more, we need to check only corners of the plane because of the property of N_{p1} and N_{p2} mentioned above.

Case 2: N_{p1} and N_{p2} may not intersect at all, resulting in one equation lying above the other in the ranges of x, y, and z. In this case, simply, two points are evaluated: ($x=0$, $y=0$, $z=0$) *and* ($x = ns_{ACD}$, $y = ns_{CD}$, $z = ns_{ABCD}$). If N_{p1} is larger than N_{p2} at $x=0$, $y=0$, *and* $z=0$, $N_{p1}(x=0, y=0, z=0)$ is the minimum solution. Otherwise, $N_{p2}(x = ns_{ACD}, y = ns_{CD}, z = ns_{ABCD})$ is the minimum solution.

The worst case of the above algorithm is when it requires checking all of the candidate points. Those candidate points can be enumerated by setting minimum or maximum values to variables except one variable. Therefore, the complexity is $O(k \cdot 2^{k-1})$ where *k* is the number of variables. In this case, *k*=3. Notice that the computational complexity of this algorithm is independent of the network size.

In order for a clustering algorithm to cluster primitive cells with the minimum number of pASIC3 logic cells, we need to know the distribution of pASIC3 logic cell types as well. Let us define n_{2A+2B} as the number of type 2A+2B pASIC3 logic cells, and n_{2A+2B} and n_{2A+C} can be defined similarly. Then the following equations compute the number of pASIC3 logic cells for each type:

$$
\begin{aligned}
&n_{A+B+D} = ns_D + y \\
&n_A = \left(ns_{AD} + x + z\right) - n_{A+B+D}, \quad n_B = \left(ns_{BCD} + ns_{ABCD} - z\right) - n_{A+B+D} \\
&n_{2A+C} = ns_{ACD} - x + ns_C + ns_{CD} - y \\
&n_A = n_A - 2n_{2A+C} \\
&n_{2A+2B} = \max\left(\left\lceil \frac{n_A}{2} \right\rceil, \left\lceil \frac{n_B}{2} \right\rceil\right)
\end{aligned}
\tag{5}
$$

3.6 Clustering

A cluster is a group of primitive cells, which can be implemented in a pASIC3 logic cell. Our solution so far does not take into account the placement and routing information. The interconnect cost must be considered carefully so as not to pack nodes, which are placed far away from each other. To address this issue, first, we perform a global placement of the mapped network comprising of the primitive cells by using a state-of-the-art placement package (i.e., DRAGON2000 [19].) The placement result is used to specify the spatial proximity of primitive cells. Next, we decide which base gate will realize each primitive cell in a deterministic order according to a solution presented. Then, we randomly pick a node, and the node becomes a seed for a cluster. Then, we try to find the closest node to this cluster. A simple algorithm checks if the node can be merged into the cluster according to the distribution of types of pASIC3 logic cells. Searching for a new node repeats until the cluster is full. This whole procedure continues until all nodes are absorbed in clusters. We implemented this simple algorithm to make sure the correctness for real circuits. We are under development of algorithms to improve performance subject to the minimum area constraint.

4. Performance-Driven Clustering

In this section, we present a labeling algorithm to minimize the number of pASIC3 logic cells on the longest input-output path and provide a clustering algorithm with slack-time relaxation.

4.1 Problem statement

Delay caused by inter-cluster interconnect, which connects pASIC3 logic cells through interconnect wires and antifuses, tends to be much larger than the delay caused by intra-cluster interconnect. Therefore, we can assume that inter-cluster delay has a unit delay while the intra-cluster delay is negligible. This assumption is reasonable because no placement and routing information is known and the inter-cluster interconnect delay is much longer than the intra-cluster interconnect delay. The performance-driven clustering problem can be stated as follows.

Problem statement: A combinational network 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 in V represents a primitive cell in the network and each edge *(u, v)* in E represents an interconnection between primitive cells *u* and *v* in the network.

Problem 4: Given a network G mapped with the library generated. We want to find a clustering solution so that the number of pASIC3 cells on the critical path is the minimum.

Notice that each cluster must be *feasible* in the sense that it must be realizable with a single a pASIC3 logic cell. label(*u*) denotes the label of node *u* in the network.

4.2 Multi-dimensional labeling algorithm

When each node has a fixed and known size, the clustering constraint is monotone, and the unit delay model is used, Lawler's algorithm guarantees the minimum number of clusters on the critical path for a combinational network [15] while no cluster size exceeds the maximum size constraint. Recall that a clustering constraint is monotone if and only if any connected subset of nodes in a feasible cluster is also feasible.

The clustering constraint for our problem is monotone as well. More precisely, when a collection of primitive cells in the mapped network can be realized in a pASIC3 logic cell, a subset of these primitive cells can also be realized in a single pASIC3 cell. We call this constraint a *resource constraint*. We propose a labeling algorithm to guarantee the optimal solution, which is subject to the resource constraint. The pseudo-code for the algorithm is provided in Figure 4. Notice that since multiple base gates can realize a node, those base gates must be checked to create a cluster. In addition, according to the topological containment described in section 3.2, some base gates are inferior to others. Therefore, inferior base gates can be dropped as was done for the area-driven clustering. This significantly reduces the complexity of generating clusters during the labeling phase.

```
1.  Algorithm Multi-dimensional labeling
2.  Begin
3.  foreach primary input v do
4.    label(v) = 0;
5.  end for;
6.  Generate list T of non-primary inputs in
7.    topological order;
8.  While T is not empty, then
9.    Remove node v from the head of T;
10.   ℓ = max{label(u) | u ∈ input(v)};
11.   clusterSet(v) = ∅;
12.   Generate list M of base gates for node v;
13.   foreach base gate from M,
14.     R = form clusters from node v and clusters with
15.      label ℓ in fanins of node v
16.     foreach cluster from R,
17.       if cluster is feasible for pASIC3 realization,
18.          clusterSet(v) = clusterSet(v) ∪ cluster;
19.       end if;
20.     end for;
21.   end for;
22.   if clusterSet(v) ≠ ∅, then
23.     label(v) = ℓ;
24.   else
25.     label(v) = ℓ + 1;
26.   end if;
27. end while;
28. End
```

Figure 4: Multi-dimensional labeling algorithm

The algorithm starts by setting all labels of primary inputs to zero. In line 12, candidate base gates for the node are found to create clusters for different base gates. In lines 13 to 21, we create all feasible clusters comprising of node *v* and its fanin nodes with label ℓ. If there exists any feasible cluster, the label of node v remains at ℓ. If no feasible cluster exists, the label is incremented by one. The total number of clusters for feasibility test in line 14 is an important factor to determine for computational complexity of the algorithm. We denote this number with *k*. In addition, up to four base-gates can fit in a pASIC3 logic cell. Therefore, if the number of fanin nodes with label ℓ in line 14 is larger than three fanins, then we will not have to generate clusters for feasibility test. We denote the number of fanin nodes with label ℓ, which is less than four, with *f*. *m* denotes the number of base gates for node v. Since there can be *f* fanin nodes with label ℓ, each of which can have up to *k* clusters, node v can choose a base gate out of *m* base gates, the total clusters for feasibility test generated in line 14 will be at most $m \times k^f$, which is independent of network size. Therefore, the complexity of this algorithm becomes $O(|V| \times m \times k^f)$, where |V| is the number of nodes.

An example is provided in Figure 5. For the sake of simplicity, only two cluster (pASIC3 logic cell) types are considered: *2A+2B*

and 2A+C. Each character annotation in a node represents a candidate base gate for realization. There can be different cluster types. Selecting the cluster type for area minimization during the merging phase is an open question. We will describe our strategy in section 4.4.

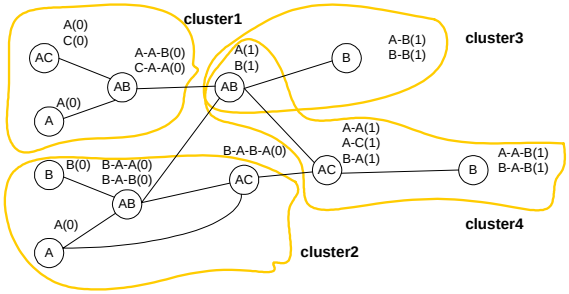

Figure 5: Clustering example.

4.3 Slack-time relaxation

The labeling algorithm generates a cluster solution where some clusters can cover identical nodes. If there is no slack-time relaxation, the nodes must be duplicated in these clusters. For example, *cluster3* and *cluster4* cover the same node. If a designer specifies a required maximum label in the primary outputs, then we can compute the slack time for each node by subtracting the label of the node from the required label at the node. A positive slack of a node denotes the amount by which the node can be slowed down without increasing the maximum label. Therefore, we can sort nodes by their descending slack values. Next, we process each node from the sorted order. If other clusters cover a fanin node of the current node and the slack of the node is positive, this will tell us that the fanin node can be removed from the cluster, which contains the node and its fanin node. By doing this, the current node will decrease its slack by one. If this operation changes the slack time of the fanin node, then the slack time of transitive fanin cone of the node is computed again. This procedure prevents unnecessary node duplication while the required maximum label is still met.

4.4 Merging algorithm

After finding all clusters in a network, we may be able to merge some of these clusters if the merging still results in a feasible cluster (one that can be mapped to a single pASIC3 logic cell.) In order to get insight about how to merge clusters, we performed a global placement of the clustered network by using DRAGON2000 [19]. We then randomly choose a seed cluster and find the closest cluster to it (one with the shortest Euclidean distance from the seed cluster.) We then attempt to merge the two clusters into one. Of course, the merged cluster must be feasible. If the merged cluster still has room in it, then we continue to look for another nearby cluster. This process is continued until the merged cluster is full. Next, we do the same expansion/merging process starting with another seed cluster. This procedure is repeated until no cluster with low area utilization is left or until no more merging is possible.

5. Experiment Results

We have selected 18 large combinational circuits from the MCNC91 benchmark. SIS [17] reads the circuits in blif format.

To evaluate our library generation and area-driven clustering, we compare our results to those from a commercial tool, called QuickWorks 4.1 from QuickLogic. For QuickWorks 4.1, the following options were selected to minimize area:

Logic optimization: level – technology map, mode-overnight, type-area, and no buffer insertion

Table 3: Results of area-driven clustering

Circuits	QuickWorks			Packer-area			Improvement (%)		
	Cell fragments	Max-depth	PASIC3 logic cells	Primitive cells	Max-depth	PASIC3 logic cells	Max-depth	Number of primitive cells	Number of pASIC3 logic cells
i9	356	9	95	384	8	96	11.11	-7.87	-1.05
rot	398	15	104	350	18	88	-20.00	12.06	15.38
i8	706	9	184	568	12	142	-33.33	19.55	22.83
pair	925	15	243	802	22	213	-46.67	13.30	12.35
vda	514	10	131	312	13	79	-30.00	39.30	39.69
x1	176	6	45	165	7	42	-16.67	6.25	6.67
C6288	1904	91	476	1513	99	448	-8.79	20.54	5.88
C5315	996	16	264	699	18	196	-12.50	29.82	25.76
alu4	500	25	125	419	35	113	-40.00	16.20	9.60
apex6	360	9	124	329	13	84	-44.44	8.61	32.26
C880	218	20	57	177	24	54	-20.00	18.81	5.26
C3540	705	23	181	672	34	175	-47.83	4.68	3.31
alu2	262	32	66	216	24	57	25.00	17.56	13.64
C1355	224	17	57	210	15	53	11.76	6.25	7.02
C1908	221	25	56	215	23	55	8.00	2.71	1.79
C432	121	25	31	120	26	31	-4.00	0.83	0.00
C499	201	13	58	210	14	53	-7.69	-4.48	8.62
Average improvement							-16.24	12.01	12.29

Placement and Route: overnight

QuickWorks uses the term *cell fragment* to indicate a library cell generated from a *pASIC3* logic cell. The results were taken after placement [16]. For our simulation set-up, the library was read and *script.rugged* was used to optimize a circuit. SIS was used for technology mapping with the library. We estimated the minimum number of logic cells by using our algorithms, PackGen-area. Table 3 reports the results of the area-driven clustering. In most of the cases, PackGen-area used a fewer primitive cells than QuickWorks. PackGen-area reduced the number of pASIC3 logic cells by 12.29% on average compared to QuickWorks. On the other hand, PackGen-area gives more depth of pASIC3 logic cells than QuickWorks for many cases. For this paper, we did not apply algorithms to minimize the depth. Various heuristic algorithms are under development. For example, picking up a seed node with high probability of reducing depth will reduce the depth than randomly selecting a seed, and so forth.

Results of the performance-driven clustering algorithm, called PackGen-delay, are provided in Table 4. QuickWorks is set to minimize delay during logic optimization. Placement and routing is also set to the overnight mode for the best result. The maximum depth of a circuit in the table is the number of pASIC3 logic cells on the longest input-output path. The comparison of the numbers of primitive cells before and after slack-time (ST) relaxation shows that our proposed method effectively avoids logic duplication. The number of clusters for each circuit was reduced after the merging phase. For this paper, we do not give the threshold of distance between placed clusters, and some far-away clusters might have been merged together. However, we can control the results by changing the threshold. As a result, QuickWorks used much more pASIC3 logic cells than the results from the area-driven clustering. Because of the logic duplication, our algorithm on average uses more pASIC3 logic cells after merging. Compared to QuickWorks,

PackGen-delay reduced the maximum depth of the circuit by 44.75% on average with a 12.05% area overhead.

6. Conclusion

In this paper, we presented area-driven and performance-driven clustering algorithms for coarse-grained, anti-fuse based FPGAs. For area-driven clustering, we set up a pair of linear equations and found the optimal solution to find the minimum number of required pASIC3 logic cells. For performance-driven clustering, we proposed a labeling algorithm so that it can generate the minimum number of clusters on the critical path. A slack-time relaxation was used to avoid redundant logic duplication without violating performance constraint. In addition, a random merging was used to cluster closely placed partially filled clusters. Experimental results showed that the area-driven clustering algorithm used fewer numbers of pASIC3 logic cells by 12.29 % on average and the performance-driven clustering algorithm reduced the maximum depth by 44.75%, on average.

References

[7] Eric Kusse, and Ran Rabaey, "Low-energy embedded FPGA architecture," in Proc. International Symposium on Low Power Electronics and Designs, pp. 155-160, 1998.

[8] Alexander Marquardt, Vaughn Betz, and Jonathan Rose, "Speed and area tradeoffs in cluster-based FPGA architecture," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 8, no. 1, February 2000, pp. 84 – 93.

[9] *pASIC3* FPGA Family Datasheet, QuickLogic Corporation (http://www.quicklogic.com).

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

[11] 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.

[12] 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.

[13] E.Bozogzadeh, S. Ogrenci-Memik, M. Sarrafzadeh, "Rpack: routability-driven packing for cluster-based FPGA," in Proc. Asia-South Pacific Design Automation Conference, pp. 629 – 634, 2001.

[14] Chang Woo Kang, Ali Iranli, and Massoud Pedram, "Technology mapping and packing for coarse-grained, anti-fuse based FPGAs," in *Proc. Asia and South Pacific Design Automation Conference*, January 2004.

[15] E. L. Lawler, K. N. Levitt, J. Turner, "Module clustering to minimize delay in digital networks," *IEEE Transactions on Computers*, vol. C-18, no. 1, January 1969, pp. 47 – 57.

[16] QuickLogic.com, QuickWorks User Manual

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

[18] Thomas H. Cormen, Charles E. Leiserson, and Ronald, L. Rivest, *Introduction to Algorithms*, McCraw-Hill Book Company, 2000.

[19] M. Wang, X. Yang, and M. Sarrafzadeh, "Dragon2000: standard-cell placement tool for large industry circuits," in *Proc. International Conferecence on Computer Aided Design*, 2000, pp. 260-263.

[20] http://mathworld.wolfram.com

Table 4: Results of performance-driven clustering

	QuickWorks 4.1			PackGen-performance						Improvement (%)	
Circuits	Number of fragment cells	Number of pASIC3 logic cells	Maximum depth	Number of primitive cells after technology mapping	Number of primitive cells after duplication		Number of clusters (pASIC3 logic cells)		Maximum depth	Number of pASIC logic cells	Maximum depth
					Before ST-relaxation	After ST-relaxation	Before merging	After merging			
i9	381	97	10	384	626	398	262	148	5	-52.58	50.00
rot	404	111	15	338	429	342	199	88	7	20.72	53.33
i8	771	201	9	607	1007	715	449	210	5	-4.48	44.44
pair	900	238	13	818	1052	822	478	231	10	2.94	23.08
vda	546	139	10	304	383	333	163	111	5	20.14	50.00
x1	193	50	6	170	193	171	88	56	3	-12.00	50.00
C6288	1872	771	82	1506	1620	1525	829	496	55	35.67	32.93
C5315	992	430	18	671	774	692	448	284	13	33.95	27.78
alu4	521	131	25	438	570	473	222	170	12	-29.77	52.00
apex6	361	112	9	330	489	334	230	91	6	18.75	33.33
C880	219	55	18	178	230	180	123	65	14	-18.18	22.22
C3540	710	179	25	654	993	693	392	198	15	-10.61	40.00
alu2	281	71	26	224	320	233	128	90	10	-26.76	61.54
C1355	223	57	14	210	312	296	138	88	7	-54.39	50.00
C1908	223	56	25	219	280	262	119	75	9	-33.93	64.00
C432	127	35	25	109	201	159	75	46	10	-31.43	60.00
C499	200	54	13	210	312	296	138	88	7	-62.96	46.15
Average Improvement										-12.05	44.75