9

Circuit Partitioning For FPGAs

G. Saucier, D. Brasen, J.P. Hiol
Institut National Polytechnique de Grenoble / CSI
46, Avenue Félix Viallet, 38031 GRENOBLE cedex FRANCE
Tel: (33) 76-57-46-87 Fax: (33) 76-50-34-21 E-mail: saucier@imag.fr

Abstract

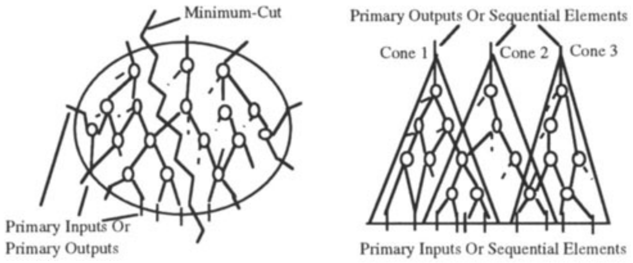
High-level synthesis tools have traditionally used circuit hierarchy to partition circuits into packages. However hierarchical partitioning can not be easily performed if hierarchical blocks have too large a size or too many IOs. This problem becomes more frequent with FPGAs which have small size limits and even smaller IO pin limits. An IO bottleneck often prevents the maximum FPGA package size from being reached. In this paper, two new FPGA cone partitioning algorithms are presented that have been implemented in the ASYL+ tool set. High-level synthesis is linked to cone partitioning by creating circuit hierarchy from a VHDL description.

1. INTRODUCTION

High-level synthesis strives to minimize the number of packages required to realize a circuit with minimum production cost at maximum frequency. The post-synthesis circuit partitioning addressed here is normally required for blocks that can not be cut based on high-level structures (architecture reorganization, data-path bit-slice partitioning, or Ram/Rom array sectioning). Such blocks are partitioned by grouping basic circuit elements like FPGA modules to meet size or IO package constraints and match the worst timing of the blocks produced during synthesis. Given that timing constraints can be met by keeping critical paths within a partition, the problem becomes finding partitions of high size to IO ratios that meet IO package constraints.

Prior work in post-synthesis circuit partitioning has primarily focused on top-down minimum-cut algorithms (Fiduccia 1982), (Hwang 1992), (Kring 1991). These algorithms recursively attempt to find the cut that minimizes the number of connections between two bipartitions (Figure 1). The min-cut of (Keringhan 1970) proposed starting with two random bipartitions followed by iterative pair swapping on all pairs of nodes. Subsequent algorithms have improved the time complexity, added a look-ahead into future moves to stabilize results, and reduced the cut size by gate replication and improved move set generation Kuznar 1993. All allow for a maximum size constraint per partition by limiting the size of a bipartition during each min-cut operation. However to meet a maximum number of IO pins constraint per partition the min-cut algorithm has to cut further, reducing the size as well as IOs of otherwise optimal partitions.Traditional min-cut approaches lack the ability to find constrained IO partitions with high Size/IO.

To meet high Size/IO requirements for specific packages, cone structure partitioning is proposed. Define a cone structure as the set of all combinational nodes that can be found between a single output and the inputs that lead to that single output (Figure 1). For sequential circuits, the inputs/outputs of the cone can also be the outputs/inputs of sequential elements. Cone partitioning selects and merges/cuts high Size/IO combinational groups within cones that exist because of low fanout. Circuits with low fanout are easily partitioned into unoverlapping cones that fit packages. Circuits with high fanout have the majority of each cone overlap several others and merging/cutting of optimal high Size/IO circuit sections is required to produce good partitions.


Minimum-Cut
Primary Inputs Or
Primary Outputs
Primary Outputs Or Sequential Elements
Cone 1
Cone 2
Cone 3
Primary Inputs Or Sequential Elements

In this paper, a hierarchical partitioning algorithm is proposed that preserves high-level timing. For those nonhierarchical blocks found to be too large, flat cone partitioning is analyzed. Critical path containment within a partition is shown to be easy with cone partitioning with no loss to area. A first cone partitioning algorithm clusters by cutting each cone around highest cost cones and then cluster merges. A second cone partitioning algorithm clusters nodes found within the same cone overlapping region and then cluster merges. Experimental results using MCNC and industrial benchmarks on two different industrial FPGA packages show that the cone partitioning algorithms produce better partitions than min-cut. The hierarchical partitioning algorithm is also used to partition circuit hierarchy created from an example VHDL description.

2. Hierarchical Partitioning Algorithm

For FPGA implementations, circuit hierarchy constrains partitioning with a prioritized list of gate clusters within clusters. Since this can guarantee the timing of high-level structures, partitioning based on circuit hierarchy can be powerful. However if the designer or high-level synthesis tool that creates the circuit hierarchy does not understand the final technology, the hierarchy overconstrains producing bad partitions with low gate utilization.

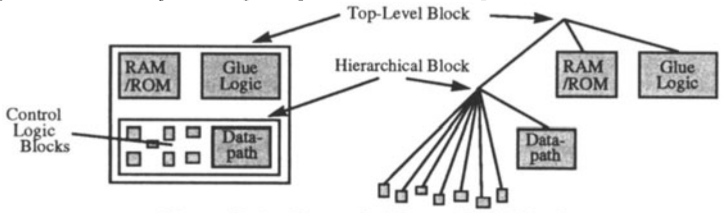


Figure 2: An Example Hierarchical Circuit.

For example, refer to the circuit hierarchy of Figure 2. As seen from a high-level, there is a Data-Path block, a RAM or ROM block, and Glue-Logic scattered as a controller for the data-path and a separate block of logic. At the post-synthesis level, the contents of all these blocks are mapped to FPGA library macros or modules which represent gates. The timing designed for at the high-level is preserved by implementing such blocks within one FPGA package. If the circuit hierarchy is flattened, a connected network of FPGA modules from an FPGA library results. No more Data-Path, RAM, or ROM since the typical FPGA currently has no specialized hardware sections. The only way to guarantee the desired high-level timing is to partition based on circuit hierarchy.

To exploit the hierarchy and preserve high-level timing, the following algorithm is proposed to perform hierarchical circuit partitioning.

Hierarchical Partitioning Algorithm:

Step 0) Begin with a pointer to the top-level block assigned as the current BLOCK.

Step 1) If the current BLOCK has no hierarchical sub-blocks AND the BLOCK is larger than Size or IO constraints, perform Flat Cone Partitioning on the BLOCK. The BLOCK will be replaced by non-hierarchical sub-blocks (i.e., flat circuit partitions that meet constraints) under a new BLOCK pointer at the same level of hierarchy.

Step 2) If there exists hierarchical sub-blocks in the current BLOCK, recursively call the hierarchical partitioning algorithm for each sub-block (i.e., set BLOCK=sub-block and goto step 1).

Step 3) Perform pair-wise merging on resulting nonhierarchical sub-blocks of BLOCK to meet constraints. Merge sub-blocks (i,j) to meet MaxSize or MaxIO constraints and minimize the number of packages. For FPGAs this means to maximize the cost ratio

$$Cost_M = \left(\frac{Size_M = Size_i + Size_j}{IO_M = IO_i + IO_j - (2 * IO_{Common})}\right)$$

since IO pin bottlenecks cause MaxSize constraints to be rarely met. IOcommon is the number of common IO connections between the blocks. The block with largest cost=Size/IO is merged with the block that will produce a new block with largest cost. Finding larger cost merges is not a problem since size grows faster than IOs. Partitions are filled sequentially allowing for the selection of different packages. When finished merging, remove this level of hierarchy and analyze the next (i.e., move all sub-blocks of BLOCK to the parent of BLOCK, set BLOCK to its parent, and goto step 1). Exit if BLOCK is the top-level block since the circuit hierarchy has been fully partitioned.

By recursively descending the circuit hierarchy, blocks smaller than constraints are first merged together. Such blocks connected only through the top-level are unlikely to be implemented in the same FPGA. The blocks larger than constraints are partitioned with cones.

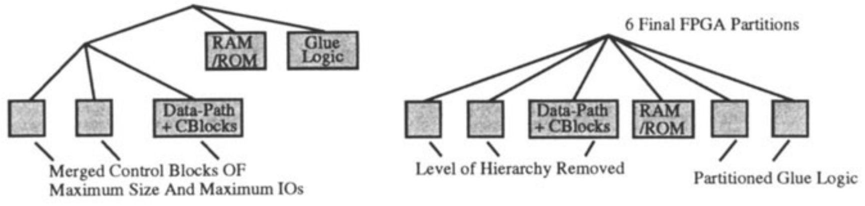


Figure 3: Partitioning The Example Hierarchy.

In Figure 3, the Hierarchical Partitioning Algorithm is performed on the example of Figure 2. The small control blocks of the bottom-level hierarchy are first merged, some with the Data-Path block, until MaxSize and MaxIO limits prevent further merging (Figure 3 - Left). The bottom-level hierarchy is removed and partitioning performed at the top-level (Figure 3 - Right). The Glue-Logic block was too large for constraints and was cone partitioned into two blocks. The Circuit Hierarchy is thus partitioned into 6 FPGAs with the original RAM/ROM and Data-Path blocks fully contained within an FPGA package to preserve high-level timing. Hierarchical partitioning in this way assumes some a priori rules during the creation of circuit hierarchy by designers or high-level synthesis tools to achieve good results.

• Since the target FPGAs are rarely known when designing at the high-level, nonhierarchical flat blocks should be made smaller than the smallest possible FPGA package considered. This facilitates the post-synthesis merging of the hierarchical partitioning when the target FPGA is known. Also the high-level structures of Data-Paths, RAMs, and ROMs are more properly partitioned at the high-level anyway.

• Merge together the hierarchical blocks which do not serve to preserve high-level timing. The flat cone partitioning algorithms to follow seek out highly connected sections and more accurately fill the target FPGAs since the block granularity used is the basic FPGA module. The local timing considerations of containing entire critical paths within a package are also performed by the flat cone partitioning.

These two rules are not in conflict since the first is a rule for blocks which should not be further partitioned and the second is for those that should. With hierarchical blocks designed as pre-partitioned circuit sections, the remainder of this paper will focus on the flat cone partitioning of blocks too large for target FPGA packages.

3. CONE STRUCTURES WITHIN A FLAT CIRCUIT

Define a cone structure as the set of all combinational nodes that can be found connected between a single output and the inputs that lead to that single output. For sequential circuits, the inputs/outputs of the cone can also be the outputs/inputs of sequential elements. In the netlist of Figure 4 there are two cones. Cone 2 has an input pin and an output pin connected to the sequential element 2. Since sequential element 2 has only connections within cone 2, these two pins can be removed from the cone 2 pin list of IOs.

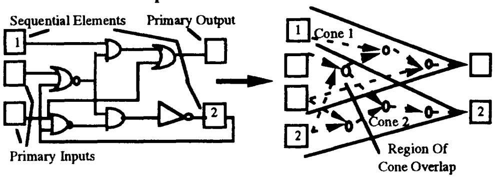


Figure 4: Finding Cone Structures Within A Flat Circuit.

Cone structures are minimum-cut structures because they outline regions of low fan-out. The only combinational gate node in Figure 4 with fanout greater than one results in an overlapping section between the two cones. A fanout greater than two would possibly result in an overlapping section with a third or fourth cone, making the cones harder to merge/cut. If all the nodes in cone 2 had fanout of one, the maximum Size/IO becomes 4/4 (4 nodes, 3 inputs, and 1 output). If all nodes in cone 2 had fanout of greater than one, the minimum Size/IO becomes 4/7 (4 nodes, 3 inputs, and 4 outputs). The difference between maximum and minimum Size/IOs becomes greater as the cone size increases.

4. CRITICAL PATH CONSIDERATIONS TO IMPROVE TIMING

Define a critical path as a long combinational path between two sequential elements or IO pads of an original unpartitioned circuit. Classical algorithms for identifying critical paths have been published (Singh 1991). If such critical paths are cut during partitioning of the circuit into packages, resulting inter-package wire capacitances cause the circuit timing to drop. Also if only the most critical path is considered during partitioning, resulting inter-package wire capacitances can be sufficiently large along secondary critical paths to cause them to become the most critical paths. Prior work has used min-cut algorithms to create initial bipartitions that contain the n-most critical paths and then not allow gates of such paths to cross bipartition boundaries. The partitioning algorithm proposed in (Murgai 1991) clusters and merges the circuit around critical paths so that the maximum delay is minimized.

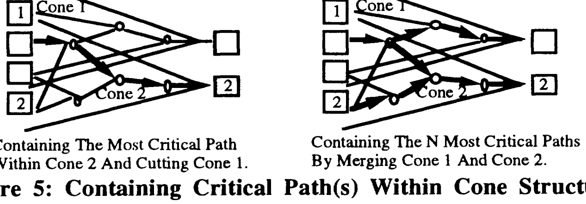

Cone 1
Cone 2
1
2
2
Containing The Most Critical Path Within Cone 2 And Cutting Cone 1.
Cone 1
Cone 2
1
2
2
Containing The N Most Critical Paths By Merging Cone 1 And Cone 2.

Figure 5: Containing Critical Path(s) Within Cone Structure.

Since cones are defined as starting from a primary output or sequential element and finishing at primary inputs or sequential elements, containing a critical path within a cone is relatively easy. On the left of Figure 5, containing the most critical path within cone 2 forces the cutting or immediate merging of cone 1. On the right of Figure 5, the n most critical paths are contained by forcing the merger of cone 1 and cone 2. Initial results indicate containing critical paths does not alter Size or IOs of partitions since most critical paths run through similar cones.

5. CUTTING FOR BEST CONES AND MERGING

In this section, the first of two cone partitioning algorithms is listed below and called *Best_Cone_Merge*. This method preserves the best cones by keeping them whole. However these whole cones are sometimes too large for package constraints with circuits of low fanout and further partitioning is required.

Cone Partitioning Algorithm 1. Best_Cone_Merge:

1. Define cone structures as described in section 3.

2. Preserve largest cost cones and cut all others. Nonoverlapping clusters result which are the highest cost sections extracted from the original circuit. Clusters that are too large for constraints, usually the whole cones that were preserved, must be divided using min-cut.

3. Add sequential elements to clusters for reduced Size/IO ratio. This means adding to clusters with the largest number of common interconnections.

4. Merge clusters with combinational/sequential elements on the same critical path as in section 4.

5. Merge clusters into partitions to meet package constraints as in the hierarchical partitioning algorithm of section 2.

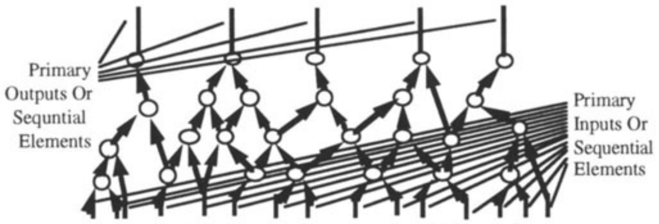

Primary Outputs Or Sequntial Elements
Primary Inputs Or Sequential Elements

Figure 6: Example Combinational Network.

To illustrate this algorithm, a combinational network example is given in Figure 6. It has a total size 26 assuming each node has a size of 1 and total number of IOs 20.

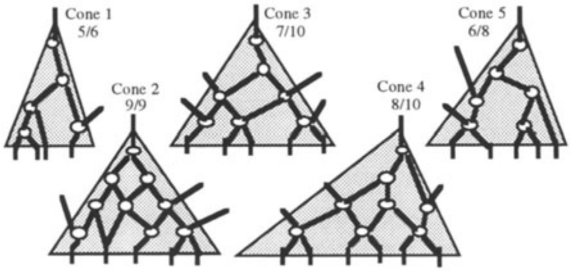


Figure 7: Cones Of Example Network And Associated Costs.

Figure 7 shows the network of Figure 6 after the definition of cones.

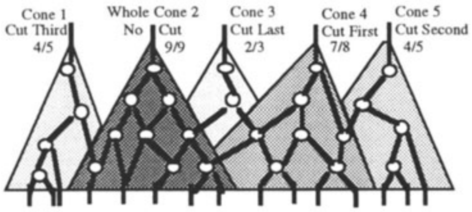


Figure 8: Flattened Cone Structure With Whole Best Cone 2.

In Figure 8, cone 2 with the greatest cost 1 remains whole. The other cones are cut around cone 2 producing cone 1 cost of 4/5, cone 3 cost of 4/7, cone 4 cost of 7/8, and cone 5 cost of 6/8. Of these cut cones, the highest cost cone is cone 4. Cones 3 and 5 are cut around cone 4.

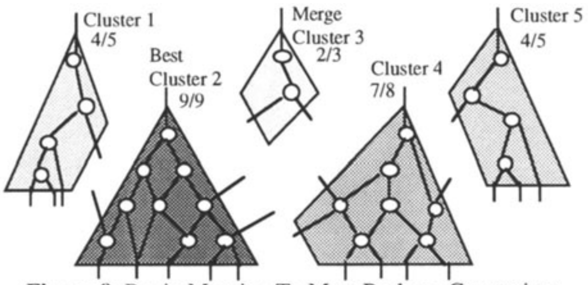


Figure 9: Begin Merging To Meet Package Constraints.

In Figure 9, clusters have resulted from the cut cone sections. Cluster 2 has the greatest cost 1. The costs after merging with cluster 2 are 13/12 for 2-1, 11/10 for 2-3, 16/15 for 2-4, and 13/14 for 2-5. The highest cost merge 2-3 is accepted if the merged cluster 2-3 meets package constraints.

6. CLUSTER ALONG CONE BOUNDARIES AND MERGE

In this section, the second cone partitioning algorithm *Cluster_And_Merge* is introduced. This algorithm clusters nodes found in similar overlapping cone sections. However these clusters often contain only one node with circuits of high fanout, forcing reliance completely on the merging operations to achieve high Size/IO partitions. Therefore this algorithm contrasts the first in that it should be used for circuits of low fanout and the first algorithm used for circuits of high fanout.

Cone Partitioning Algorithm 2. Cluster_And_Merge:

1. Define cone structures as described in section 3.

2. Cluster Overlapping Cone Sections. Nonoverlapping clusters result which are the highest cost sections extracted from the original circuit.

3. Add sequential elements to clusters for reduced Size/IO ratio.

4. Merge clusters with combinational/sequential elements on the same critical path.

5. Merge clusters into partitions to meet package constraints as in cone partitioning algorithm 1.

Figure 10 illustrates overlapping regions for the example network of Figure 6. The degree of shading in a region reflects the amount of overlap between cones. The dark region at the center is the overlap region of the three cones 2, 3, and 4. The node within this dark region has the three labels 2, 3, and 4.

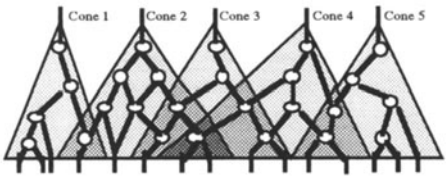


Figure 10: Defining Overlapping Region Clusters For Example Network.

All combinational gate nodes with the same set of labels are contained within the same overlapping region and are assigned to the same cluster in Figure 11. Note that the node at the top of clusters which were overlapping cone regions always have a fanout greater than one.

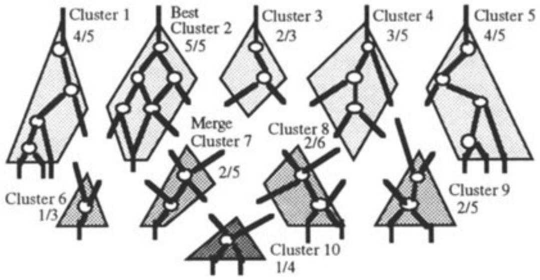


Figure 11: Block Clusters Formed From Cone Cuts.

In Figure 11, cluster 2 has largest cost 1. The merge costs after merging with cluster 2 are 9/10 for 2-1, 7/8 for 2-3, 8/10 for 2-4, 9/10 for 2-5, 6/6 for 2-6, 7/6 for 2-7, 7/11 for 2-8, 7/10 for 2-9, and 6/9 for 2-10. Cluster 2-7 is merged if it meets package constraints.

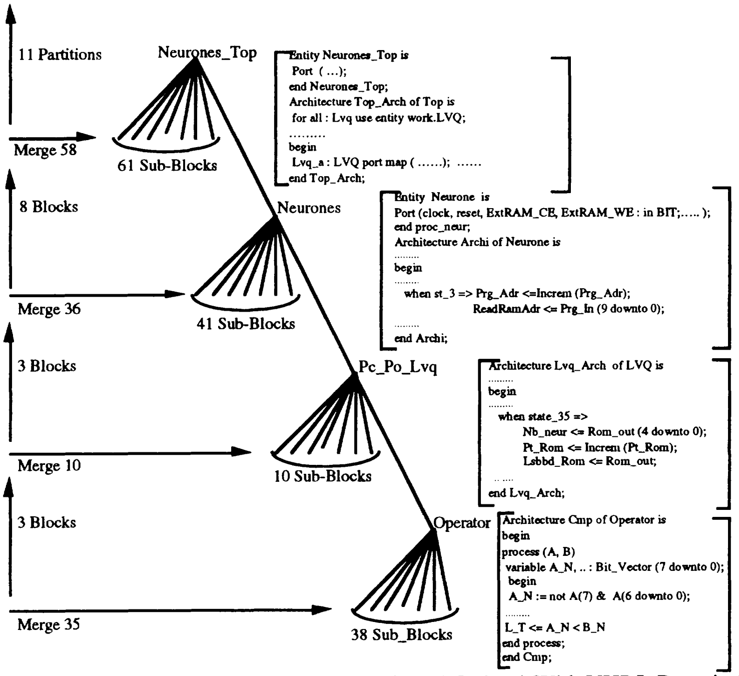
7. Experimental Results

Table 1 lists cone partitioning information for ISCAS85 MCNC benchmarks and some real industrial circuit benchmarks. Size is the number of ACTEL2 FPGA logic modules. Primary inputs and outputs are of the initial circuit. Benchmarks with many more algorithm 2 clusters than cones indicate high cone overlap and circuits with high fanout. Cone structures are rooted at the input nodes of primary outputs or sequential element inputs. Thus for combinational circuits there can never be more cones than primary outputs, but there can be more primary outputs than cones. Since sequential elements have multiple inputs, the same does not apply.

	Total	Primary	Primary	Whole	Sequential	Alg.1	Alg.2
Circuits	Size	Inputs	Outputs	Cones	Elem Size	Clusters	Clusters
c7552	9425	207	108	107	0	107	251
c5315	7005	178	123	123	0	123	284
c3540	5712	50	22	22	0	22	112
c6288	3216	32	32	32	0	32	62
c2670	3191	233	140	64	0	64	119
c880	1096	60	26	26	0	26	39
c499	912	41	32	32	0	32	43
Indust1	7362	21	92	102	171	102	195
Indust2	1287	41	96	96	0	96	101
Indust3	928	38	25	63	35	63	119

Table 1: Benchmark Circuit Information.

The industrial benchmark indust1 was synthesized from a behavior level VHDL description. During the high-level synthesis process, a netlist hierarchy was created with the partitioning rules stated. Figure 12 lists a VHDL description summary of each hierarchical block and at each level of hierarchy. During the netlist partitioning, the hierarchical blocks were easily



```
Entity Neurones_Top is
 Port ( ...);
end Neurones_Top;
Architecture Top_Arch of Top is
 for all : Lvq use entity work.LVQ;
.........
begin
 Lvq_a : LVQ port map ( ......);  ......
end Top_Arch;
```

```
Entity  Neurone  is
Port (clock, reset, ExtRAM_CE, ExtRAM_WE : in BIT;..... );
end proc_neur;
Architecture Archi of Neurone is
.........
begin
.........
  when st_3 => Prg_Adr <=Increm (Prg_Adr);
               ReadRamAdr <= Prg_In (9 downto 0);
.........
end Archi;
```

```
Architecture Lvq_Arch  of LVQ is
.........
begin
.........
  when state_35 =>
        Nb_neur <= Rom_out (4 downto 0);
        Pt_Rom <= Increm (Pt_Rom);
        Lsbbd_Rom <= Rom_out;
  .. ....
end Lvq_Arch;
```

```
Architecture Cmp of Operator is
begin
process (A, B)
 variable A_N, .. : Bit_Vector (7 downto 0);
  begin
  A_N := not A(7) &  A(6 downto 0);
 .........
 L_T <= A_N < B_N
end process;
end Cmp;
```

Figure 12: Hierarchical Partitioning Of Benchmark Indust1 With VHDL Descriptions.

merged to a minimum number of 11 packages with Size constraint 684 and IO constraint 104. This optimal result (i.e., 11 packages * 684 = 7524 is just greater than circuit size 7362) would not have been possible if the high-level synthesis had not created highly connected blocks which were smaller than the package constraints.

In Tables 2 and 3, experimental comparisons with a min-cut algorithm of (Fiduccia 1982) are presented against the cone partitioning algorithm1 of *Best_Cone_Merge* and the cone partitioning algorithm 2 of *Cluster_And_Merge*. Benchmark circuits were partitioned for two industrial packages. Table 3 lists partitioning information for a smaller package with maximum size constraint of 684 and maximum number of IO pins constraint of 104. Table 4 lists partitioning information for a larger package with maximum size constraint of 1232 and maximum number of IO pins constraint of 140. For each algorithm, Tables 2 and 3 list the number of packages into which the circuit has been partitioned (**#Pkg**), the average size of the circuit partitions (**AvSize**), the average number of IO pins of the circuit partitions (**AvIO**), and the CPU time in minutes taken during the execution of the algorithm on a Sun Sparcstation1+ (**Time**). Min-cut consistently produced bad partitions resulting in more packages because of the FPGA package IO bottleneck. Differences in results between the cone partitioning algorithms is not as clear since the typical circuits used here have average fanout (i.e, algorithm 1 clusters all met package constraints and algorithm 2 clusters all had size greater than 1).

	Min-Cut				Alg 1. Best Cone Merge				Alg 2. Cluster And Merge			
Circuits	#Pkg	AvSize	AvIO	Time	#Pkg	AvSize	AvIO	Time	#Pkg	AvSize	AvIO	Time
c7755	19	497	83	27.8	15	628	67	4.9	14	674	89	6.4
c5315	24	292	71	25.7	11	636	60	4.1	11	637	64	4.2
c3540	13	440	70	8.6	10	571	91	2.6	9	635	75	1.1
c6288	9	357	70	22.9	6	536	78	5.8	5	643	92	2.8
c2670	8	399	79	6.9	6	532	50	1.5	5	639	90	1.0
c880	3	274	93	0.1	2	548	78	0.1	2	548	81	0.1
c499	4	228	78	0.1	2	456	52	5.5	2	456	55	0.1
Indust1	17	430	78	32.1	11	669	75	9.7	11	670	88	6.3
Indust2	4	322	92	1.8	2	644	89	1.2	3	429	83	0.3
Indust3	4	232	67	1.4	3	309	86	0.6	2	464	98	1.1

Table 2: Results With Package Constraints Max Size=684 And Max IOs=104.

	Min-Cut				Alg 1. Best Cone Merge				Alg 2. Cluster And Merge			
Circuits	#Pkg	AvSize	AvIO	Time	#Pkg	AvSize	AvIO	Time	#Pkg	AvSize	AvIO	Time
c7755	13	725	96	16.2	8	1178	119	5.2	9	1048	110	8.9
c5315	11	637	121	10.6	6	1168	108	6.3	6	1168	90	6.0
c3540	8	714	79	3.3	5	1143	96	2.8	5	1143	91	1.4
c6288	5	644	87	8.4	4	804	83	7.4	3	1072	99	6.2
c2670	5	639	103	4.5	3	1064	101	1.7	3	1064	119	2.0
Indust1	10	737	86	14.2	6	1227	86	10.8	6	1227	99	7.1
Indust2	2	644	117	0.8	2	644	74	1.6	2	644	104	0.3

Table 3: Results With Package Constraints Max Size=1232 And Max IOs=140.

The graphs in Figures 13 and 14 illustrate the comparisons between min-cut and the cone partitioning algorithms for the smaller and larger packages. AvSize/AvIOs (i.e., average size over average number of IO pins) are graphed against the number of partitioned packages for each benchmark circuit. Note that the AvSize/AvIOs with cone partitioning are always greater than min-cut. Also most cone partitioning AvSize/AvIOs are greater than the small package Size/IO constraint of (684/104)=6.57 and the large package Size/IO constraint of (1232/140)=8.8 (see the vertical lines shown in Figures 13 and 14). Most min-cut AvSize/AvIOs are less than both large and small package Size/IO constraints.

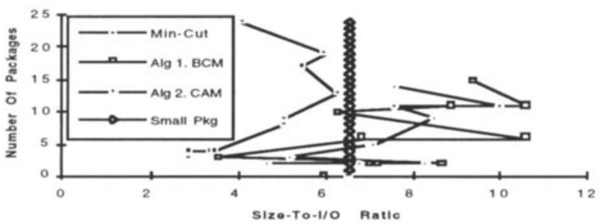


Figure 13: Cone Partitioning/Min-cut Results For Small Package.

Averaging the AvSize/AvIOs with the smaller package for the min-cut algorithm gives 4.4, for the *Best_Cone_Merge* algorithm gives 7.9, and for the *Cluster_And_Merge* algorithm gives 7.3. Averaging the AvSize/AvIOs with the larger package for the min-cut algorithm gives 7.0, for the *Best_Cone_Merge* algorithm gives 10.8, and for the *Cluster_And_Merge* algorithm gives 10.5. The difference between the min-cut and *Best_Cone_Merge* algorithms for the small package is 3.5 while for the large package is 3.8.

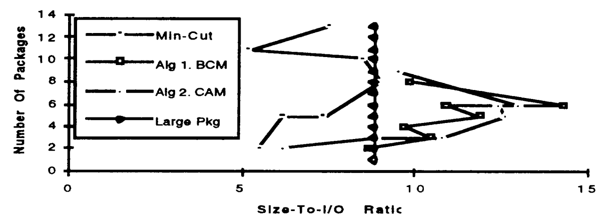


Figure 14: Cone Partitioning/Min-cut Results For Large Package.

The difference between the min-cut and *Cluster_And_Merge* algorithms for the small package is 2.9 while for the large package is 3.5. As the packages get larger with higher Size/IO constraints, the cone partitioning algorithms become better solutions for circuit partitioning.

CONCLUSION

A hierarchical partitioning and two flat cone partitioning algorithms have been presented. The first cone partitioning algorithm, which should be used with circuits of high fanout, clusters by cutting each cone around the best cost cones and then merges clusters. The second cone partitioning algorithm, which should be used with circuits of low fanout, clusters nodes found within the same overlapping cone region and then merges clusters. Experimental results using two differing industrial packages show that the cone partitioning algorithms produce better partitions than min-cut. Hierarchical partitioning with a circuit hierarchy created from a VHDL description also produced an optimal number of packages.

REFERENCES

C.M. Fiduccia et al. 1982, "A Linear-Time Heuristic For Improving Network Partitions", 19th DAC, 1982, pp. 175-181.

J. Hwang et al 1992, "Optimal Replication For Min-Cut Partitioning", ICCAD, 1992, pp.432-435.

B.W. Kernighan et al. 1970, "An Efficient Heuristic Procedure For Partitioning Graphs", Bell System Technical Journal, vol. 49, Feb. 1970, pp. 291-307.

C. Kring et al. 1991, "A Cell-Replicating Approach To Mincut-Based Circuit Partitioning", ICCAD, 1991, pp. 2-5.

B. Krishnamurthy 1993, "An Improved Min-Cut Algorithm For Partitioning VLSI Networks", IEEE Trans. On CAD, vol. CAD-33, no. 5, May 1984, pp.438-446.

R. Kuznar et al. 1993, "Partitioning Digital Circuits For Implementation In Multiple FPGA ICs", Technical Report TR93-03, MCNC, Research Triangle Park, NC, March 1993.

K.J. Singh et al. 1990, "A Heuristic Algorithm For The Fanout Problem", 27th DAC, 1990, pp. 357-360.

R. Murgai et al. 1991, "On Clustering For Minimum Delay/Area", 28th DAC, 1991, pp. 6-9.