Proc. Estonian Acad. Sci. Engng., 1997, 3, 4, 283–292
https://doi.org/10.3176/eng.1997.4.05

CIRCUIT PARTITIONING METHOD FOR FPGAs

Helena KRUPNOVA and Gabrièle SAUCIER

Institut National Polytechnique de Grenoble/CSI 46, Avenue Felix Viallet, 38031 GRENOBLE cedex FRANCE; e-mail: {bogushev,saucier}@imag.fr

Received 30 May 1997, in revised form 20 October 1997

Abstract. Since the complexity of large electronic systems is rapidly increasing, most of them do not fit in any of the commercially available Field Programmable Gate Array (FPGA) devices. Thus, the implementation of circuits with FPGAs commonly requires partitioning to obtain several FPGA packages. This paper describes an automatic partitioning method of a design onto multiple FPGA devices. The partitioning method described consists of a hierarchy-based clustering algorithm, a constructive partitioning algorithm, and an iterative improvement algorithm. Experimental results were obtained on large industrial hierarchical circuits, using XILINX 4000 technology. Experimental results show that the hierarchy-based clustering is the key to speed up the partitioning of large industrial circuits and to improve partitioning results.

Key words: partitioning, prototyping, field programmable gate array, application specific integrated circuit, clustering, flat circuit, hierarchical circuit, constructive algorithm, iterative improvement algorithm.

1. INTRODUCTION

The partitioning problem of field programmable gate arrays arises when prototyping large application specific integrated circuits (ASICs): their size is commonly too large to allow an implementation on a single FPGA device. Multiple emulation and prototyping systems use partitioning to implement the complex circuits on FPGAs. Circuit partitioning on FPGAs assumes the definition of a number of blocks, allowing for each block to satisfy a set of constraints: area, IO pin, flip-flop, and some other technology-dependent constraints. The problem is complicated because the gate capacity of FPGA chips is large with respect to their IO pin capacity. Thus, because of a rapid pin saturation, it is difficult to obtain good device fillings.

The classical partitioning objective is to minimize the number of cut nets between two or more blocks of partition. The problem is NP-complete, and heuristic methods are used to reach a solution [1–3]. Surveys about the state of art in partitioning may be found in [4, 5].

Most of the existing partitioning approaches [6–8] are based on the well-known iterative improvement heuristic proposed in [1]. But the iterative improvement algorithms are known to trap usually in a local minimum. The larger the circuit, the smaller the probability to find a global minimum. To cope with this problem, multiple approaches of clustering – grouping of the circuit elements before the partitioning starts – were proposed to reduce the complexity. However, bottom-up clustering methods [9–11] suffer from the absence of the global circuit view, and top-down clustering approaches [12] still are faced with the same complexity problem. In [13] an algorithm is proposed which combines the top-down and bottom-up clustering approaches and allows a rapid partitioning of FPGA netlists of 100 K gates. But the reported FPGA gate utilization is quite low – less than 40%. This algorithm, like other approaches, works on flat netlists at the gate level. Although the hierarchical representation is a natural basis for the reduction of complexity, few device partitioning methods for hierarchical circuits have been reported [14]. On the other hand, complex digital circuits designed manually and assisted by high-level synthesis tools are hierarchical. Therefore, this hierarchy can be used for guiding the partitioning to obtain a faster and better solution. The partitioning method presented in this paper combines hierarchical clustering [15] and the flat partitioning approach to minimize the number of obtained blocks.

This paper is organized into eight sections. Section 2 provides the statement of the partitioning problem. Section 3 covers the hierarchy-driven clustering approach to reduce the complexity of the partitioning problem. Section 4 describes the flat netlist partitioning method, consisting of the constructive and iterative improvement algorithms. Sections 5 and 6 present and discuss the experimental results obtained on a set of industrial benchmarks using XILINX 4000 technology. In section 7, our conclusions are drawn.

2. STATEMENT OF THE FPGA PARTITIONING PROBLEM

A mapped digital circuit Y is a network of primitive cells, characterized by its size S_Y and its input/output number T_Y. The size of the circuit is the sum of the sizes of all its primitive cells (such as standard cells, FPGA CLBs, flip-flops, RAMs, ROMs). The circuit may be modelled as a hypergraph $H = (V, L)$, where V is a set of nodes representing circuit cells, and L is a set of hyperedges which represents a set of nets. *FPGA partitioning problem* consists in finding a partition of the initial circuit into a minimal number of blocks b: $(A_1, A_2, ..., A_b)$, each of which can be implemented on a single target FPGA device D, characterized by its size S_{max} and IO pin number T_{max}.

3. HIERARCHY-BASED CLUSTERING APPROACH

Circuit hierarchy is defined by the hierarchy tree. Each node of the tree corresponds to a block in the circuit hierarchy. Leaf blocks contain only primitive cells. Non-leaf blocks contain other blocks and possibly primitive (i.e. glue logic) cells. Each block, or an envelope, E is characterized by its size S_E and IO number T_E. The amount of the glue logic, contained in an envelope, is denoted by G_E. An envelope may be removed, as a result, its border becomes transparent.

Clusters are defined as *hard* envelopes, which cannot be removed during partitioning. An envelope E is *acceptable* if it satisfies two conditions: $S_E \le S_{\max}$ and $T_E \le T_{\max}$. Circuit *clustering* is the process of defining a set P of hard acceptable envelopes. Clusters are the first acceptable envelopes when going from the root to a leaf in the hierarchy tree. Once the clusters are selected, the rest of envelopes will be removed. Then the flat partitioning algorithm is applied to the network which contains the glue logic cells and clusters.

The non-acceptable envelopes are either removed or split to obtain several acceptable envelopes. The strategy chosen is based on the envelope *ST*-quality (Size-Terminals quality) defined

$$ST_E = 1/2\left(AvST_E + AbsST_E\right),$$

where $AbsST_E$ is the absolute *ST*-quality calculated as $AbsST_E = S_E/T_E$; $AvST_E$ is the average *ST*-quality calculated as $AvST_E = 1/n\sum_{j=1}^{n} ST\left(C_{Ej}\right)$, where $ST(C_{Ej})$ is the *ST*-quality of the j-th successor of E. For a leaf envelope, $ST_E = AbsST_E$. Each FPGA device D may also be characterized by the absolute *ST*-quality ratio: $AbsST_D = S_{\max}/T_{\max}$. Partitioning strategy selection for non-acceptable envelopes is shown in Fig. 1.

Envelope \ ST-quality	$ST_E < AbsST_D$	$ST_E >= AbsST_D$	
		$ST_E >= AvST_Y$	$ST_E < AvST_Y$
Leaf envelopes	Remove	Split	Split
Non-leaf envelopes with $G_E > S_{MAX}$	Remove	Split	Split
Non-leaf envelopes with $G_E < S_{MAX}$	Remove	Split	Remove

Fig. 1. Selection of a strategy for envelope partitioning.

4. PARTITIONING ALGORITHM

Once the clustering is performed, the clustered circuit will be represented by a cluster graph, and the flat partitioning algorithm starts. The partitioning algorithm we applied for flat netlists is similar to the PP algorithm described in [8,16]. The algorithm is iterative: the first iteration applies a bipartitioning

procedure to the given circuit, and the subsequent iterations apply the same procedure to the remainder. At each moment, $b - 1$ blocks satisfy the target device constraints. One block called the "remainder" does not satisfy the device constraints. The recursion stops when the "remainder" meets the device constraints. Each partitioning step consists of the application of two algorithms:

1) creation of an initial partition by a constructive algorithm;

2) amelioration of the existing partition by an iterative improvement algorithm.

It has been observed that starting the bipartitioning procedure from a randomly chosen bipartition tends to produce unsatisfactory results. Because of this, we have used the combination of both constructive and iterative improvement algorithms.

4.1. Constructive Partitioning Algorithm

A greedy-like algorithm is used to create an initial bi-partitioning solution. Blocks are created by greedy merging clusters around two seed points (Fig. 2). The first seed point is the largest cluster, and the second seed point is the cluster with the maximal distance from the first one.

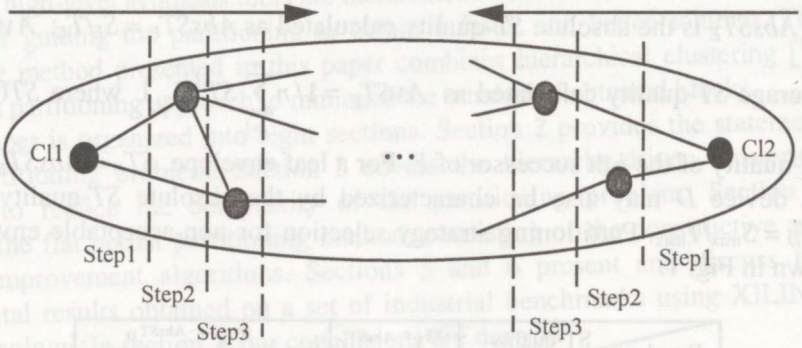

Fig. 2. Merge of clusters.

Once the candidate for merging is selected, the objective will be to ensure that the block size increases quicker than the number of IOs. The cost function to be maximized is

$$Cost_{C1+C2} = \frac{S_{C2} + S_{C1}}{T_{C2} + T_{C1} - 2 \times T_{common}}, \quad (1)$$

where S_{C1} and S_{C2} are sizes of clusters $C1$ and $C2$; T_{C1} and T_{C2} are the number of IOs of clusters $C1$ and $C2$; T_{common} is the number of connections between $C1$ and $C2$, which does not cross the cluster border when $C1$ and $C2$ are merged.

The nominator presents a good FPGA fill objective, and the denominator presents the IO number reduce objective. Block growth stops when no one cluster can be added to a block without violating the constraints. The merge continues with the second block. When both blocks are saturated, the rest of clusters are merged with the smallest block. The obtained partitioning solution is recorded.

The theoretical time complexity of the merge (if there exists an edge between each pair of clusters in the cluster graph) is $O(c^3)$, where c is the number of clusters in the network. However, in practice, cluster graphs are not dense. The practical time complexity is reduced to $O(cm^2)$, where m is the average number of edges incident to every graph node, commonly much smaller than c.

To increase the probability of the quality improvement of the initial partition, we used an additional pass with the "ratio-cut" cost function described in [12]

$$R_{(A1,A2)} = \frac{T_{A1} + T_{A2}}{S_{A1} \times S_{A2}}, \qquad (2)$$

where (A_1, A_2) is the partition in blocks A_1 and A_2; T_{A1} and T_{A2} are the number of IOs of blocks A_1 and A_2; S_{A1} and S_{A2} are the sizes of blocks A_1 and A_2. This cost function prefers partitions with small cut-set and balanced sizes.

4.2. Iterative improvement algorithm

The iterative improvement algorithm proceeds in the following four steps (Fig. 3):

1) moves to balance block sizes;

2) moves to decrease IOs on all blocks;

3) moves to decrease IOs on the "remainder" block;

4) greedy moves – only the moves which decrease the cost function are allowed.

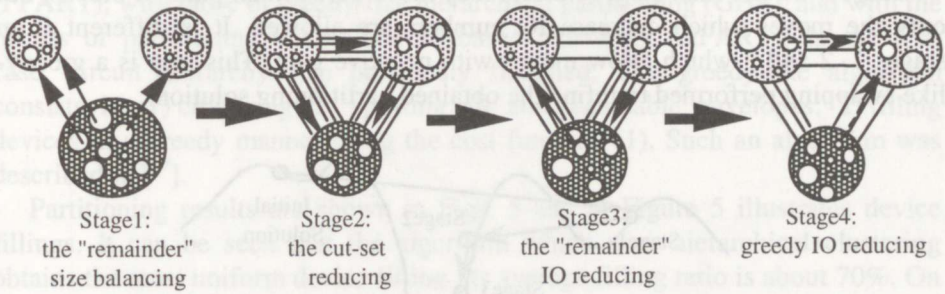

Fig. 3. Swapping steps.

Stage1. It may happen that the "remainder" block does not meet the size constraint while the minimal theoretical number M is already reached. In this

case, the size constraint is critical. The ratio-cut method of [12] suits for size balancing. The following "ratio" cost function can be used:

$$\Delta R_{(A1-C,A2+C)} = \frac{T_{A1}+T_{A2}}{S_{A1}\times S_{A2}} - \frac{T_{A1-C}+T_{A2+C}}{S_{A1-C}\times S_{A2+C}}, \quad (3)$$

where $(A_1 - C, A_2 + C)$ is the partition obtained from the partition (A_1, A_2) by moving cluster C from A_1 to A_2. This cost function reflects a decrease in the ratio value if the cluster is moved. Clusters are preferably moved "from" the largest block, which contributes to size balancing.

The drawback of this cost function is that it cannot be represented by integer values, as the net gains described in [1]. Therefore, the gain update by simple increment/decrement is not possible. The recomputing of all gain values should be performed every time the cluster is moved. The complexity of the algorithm will be $O(c^2)$, where c is the number of clusters in the network. But considering that the size of the problem was previously reduced by clustering, such an algorithm may be successfully used in practice.

Stage2. When the size constraint is satisfied for all partition blocks, but the "remainder" block does not respect the IO pin constraint, the classical IO gain cost function may be used. The classical algorithm of [3] with linear complexity is used to reduce the total number of nets in the cut-set. Several passes are performed. At each pass, all clusters are moved, and the best encountered partitioning solution is stored. We have also performed passes starting from the second best solution.

Stage3. On the previous stages, the clusters are allowed to move between all pairs of blocks – to explore the solution space in the global way. On the current stage, clusters are allowed to move only between the "remainder" block and other blocks. The solution space is explored locally to reduce the IOs number of only one block.

Stage4. On the final stage, clusters may move between all pairs of blocks, but only the moves which decrease IO numbers are allowed. It is different from stages 1, 2 and 3 which allow moves with negative gain. This step is a greedy-like swapping performed to refine the obtained partitioning solution.

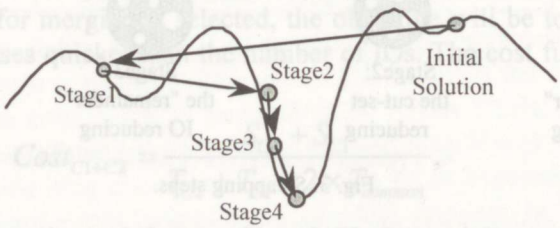

Fig. 4. Solution of cost function.

Intuitively, this process can be represented as in Fig. 4. The constructive algorithm obtains initial solution in the proximity of a local minimum. Stages 1 and 2 of the iterative improvement algorithm may be able to escape from the local minima. Stages 3 and 4 perform solution refinement by descending as low as possible in the current minimum.

5. EXPERIMENTAL RESULTS

We performed tests with four large industrial ASIC circuits migrated on XILINX 4000 technology. Benchmark data are shown in the table.

Benchmark information

Circuit	#gates before migration	Size, CLB after migration	#IO	#FF
C1	46003	3699	88	5160
C2	36133	1477	212	1874
C3	73320	7648	329	10387
C4	134532	9648	182	10165

For each benchmark, tests were performed with three different XILINX 4000 devices: XC4010 with fill 1.0 ($S_{max} = 400$, $T_{max} = 160$), XC4013 with fill 0.8 ($S_{max} = 460$, $T_{max} = 192$), and XC4013 with fill 1.0 ($S_{max} = 576$, $T_{max} = 192$). They are denoted by PKG1, PKG2 and PKG3, respectively (Figs. 5 and 6).

We compared the results of the partitioning algorithm with a flat netlist (FPART), with those of greedy-like hierarchical partitioning (GHP), and with the results of partitioning with hierarchy-based clustering (HPART). In the first case, circuit hierarchy was previously flattened. The greedy-like algorithm consists in: 1) clustering by splitting all non-acceptable envelopes; 2) filling devices in a greedy manner using the cost function (1). Such an algorithm was described in [17].

Partitioning results are shown in Figs. 5 and 6. Figure 5 illustrates device fillings. It can be seen that the algorithm which uses hierarchical clustering obtains the most uniform device filling. Its average filling ratio is about 70%. On the contrary, the filling ratio of other algorithms is not stable. Figure 6 shows the CPU results relative to HPART results. We can see that hierarchy-based clustering accelerates partitioning up to thirty times.

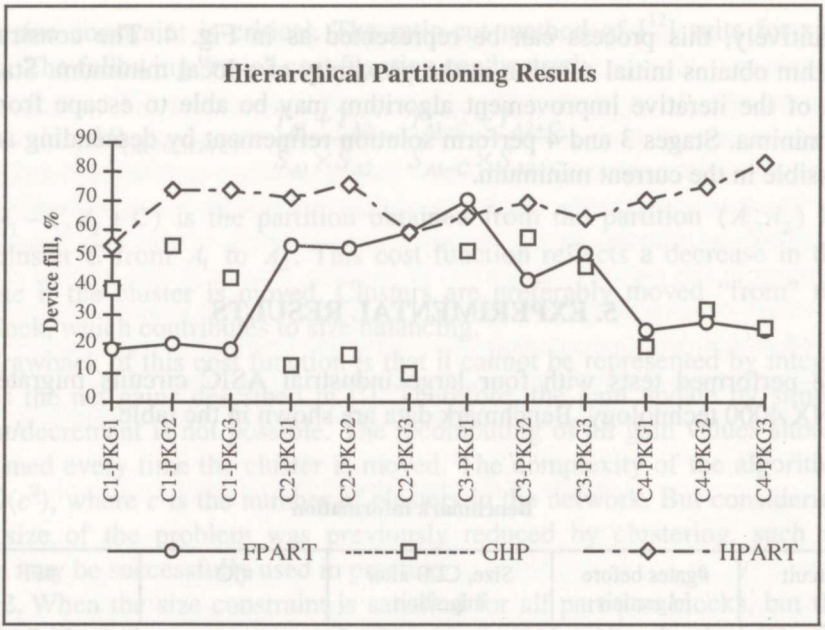

Fig. 5. Filling results of a hierarchical netlist partitioning device.

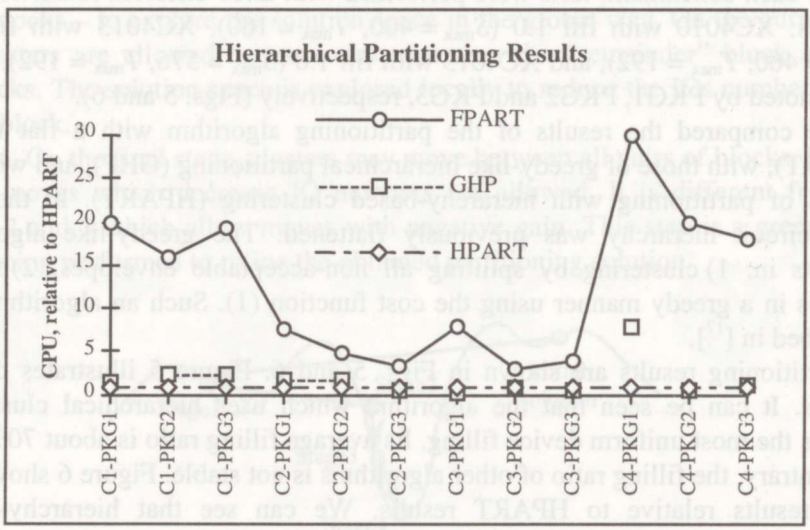

Fig. 6. CPU results of hierarchical netlist partitioning.

6. DISCUSSION

The introduced stages of the iterative improvement algorithm have different time complexity. Application of each stage for large circuits may take much of CPU time. The partitioning tool should facilitate quick acquisition of the first result. Then, it is useful to organize the work of the partitioning tool in several "passes", each of which will execute some subset of the stages. The first pass should apply only the stages with low time complexity, allowing for quick acquisition of results even for large circuits.

The example circuits are mapped on XILINX 4000 technology. To illustrate the size of the partitioning problem, it is often not sufficient to estimate the number of CLBs. The migrated netlists used were not packed, thus, one CLB does not equal to one node in the partitioning algorithm. One CLB may consist of several primitive elements, packed by the PPR tool after partitioning. In our experience, to obtain the real size of the partitioning problem, it is necessary to multiply the number of CLBs by a coefficient of 3–6.

The results of hierarchical partitioning may depend on circuit hierarchy. For example, if there are many acceptable envelopes with bad *ST*-quality which do not fit together. The method may be easily extended to treat this case.

7. CONCLUSIONS

In this paper, we have presented a design partitioning method for FPGAs. The method consists of the hierarchy-driven clustering algorithm and the flat partitioning algorithm, which, in turn, is composed of constructive and iterative improvement steps. The hierarchy-based clustering considerably reduces the size of the partitioning problem. In addition, an efficient implementation of constructive and iterative improvement algorithms allows for achieving good partitioning results, confirmed by our experimental results obtained on large industrial circuits.

REFERENCES

1. Fiduccia, C. M. and Mattheyses, R. M. A linear time heuristic for improving network partitions. *Proc. 19th Design Automation Conference*, Las Vegas, USA, 1982, 175–181.
2. Krishnamurthy, B. An improved min-cut algorithm for partitioning VLSI networks. *IEEE Trans. Computer-Aided Design of ICs and Systems*, 1984, 33/5, 438–446.
3. Sanchis, L. Multiple-way network partitioning. *IEEE Trans. Comput.*, 1989, 38/1, 62–81.
4. Alpert, C. J. and Kahng, A. B. Recent directions in netlist partitioning: A survey. *INTEGRATION, VLSI Journal*, 1995, 19, 1–81.
5. Johannes, F. M. Partitioning of VLSI circuits and systems. *Proc. 33rd Design Automation Conference*, Las Vegas, USA, 1996, 83–87.

6. Yeh, C. W. and Cheng, C.-K. A general purpose multiple way partitioning algorithm. *Proc. 28th Design Automation Conference*, San Francisco, USA, 1991, 421–426.
7. Hwang, J. and Gamal, E. Optimal replication for min-cut partitioning. *IEEE Trans. Computer-Aided Design of ICs and Systems*, 1995, 14/1, 96–106.
8. Kuznar, R., Brglez, F., and Zajc, B. Multi-way netlist partitioning into heterogeneous FPGAs and minimization of total device cost and interconnect. *Proc. 30th Design Automation Conference*, Dallas, USA, 1993, 238–243.
9. Ng, T. K., Oldfield, J., and Pitchumani, V. Improvements of a min-cut partition algorithm. *Proc. Int. Conf. Computer-Aided Design*, Santa Clara, USA, 1987, 470–473.
10. Bui, T. N., Heigham, C., Jones, C., and Leighton, T. Improving the performance of the Kernigan-Lin and simulated annealing graph bisection algorithms. *Proc. 26th Design Automation Conference*, Las Vegas, USA, 1989, 775–778.
11. Garbers, J., Promel, H. J., and Steger, A. Finding clusters in VLSI circuits. *Proc. Int. Conf. Computer-Aided Design*, Santa Clara, USA, 1990, 520–523.
12. Cheng, C-K. and Wei, Y.-C. A. An improved two-way partitioning algorithm with stable performance. *IEEE Trans.Computer-Aided Design of ICs and Systems*, 1991, 10/12, 1502–1511.
13. Chou, N.-C., Liu, L.-T., Cheng, C.-K., Dai, W.-L., and Lindelof, R. Circuit partitioning for huge logic emulation systems. *Proc. 31st Design Automation Conference*, San Diego, USA, 1994, 244–249.
14. Behrens, D., Harbich, K., and Barke, E. Hierarchical partitioning. *Proc. Int. Conf. Computer-Aided Design,* San Jose, USA, 1996, 470–477.
15. Krupnova, H., Abbara, A., and Saucier, G. A hierarchy-driven FPGA partitioning method. *Proc. 34th Design Automation Conference*, Anaheim, USA, 1997.
16. Kuznar, R., Brglez, F., and Zajc, B. PROP: A recursive paradigm for area-efficient and performance oriented partitioning of large FPGA netlists. *Proc. Int. Conf. Computer-Aided Design*, San Jose, USA, 1995, 644–649.
17. Brasen, D. Prototypage Automatique d'ASIC Sur FPGAs. *PhD Thesis, Institut National Polytechnique de Grenoble.* Grenoble, France, 1995.

DISAINI TÜKELDAMISE MEETOD PROGRAMMEERITAVATE VENTIILMAATRIKSITE TARVIS

Helena KRUPNOVA ja Gabrièle SAUCIER

Viimasel ajal on suured elektroonikasüsteemid muutunud keerukaks ja enamik neist ei mahu enam ühtegi saadaolevasse kasutajaprogrammeeritavasse ventiilmaatriksi seadmesse. Seetõttu tuleb kasutajaprogrammeeritavates ventiilmaatriksites realiseeritavate disainide korral rakendada tükeldust, et saada mitmeid väiksemaid pakette. On kirjeldatud disaini mitmesse kasutajaprogrammeeritavasse ventiilmaatriksisse automaatse tükeldamise meetodit. Tükeldamisalgoritm koosneb hierarhial põhinevast klasterdusalgoritmist, konstruktiivse tükeldamise algoritmist ja iteratiivsest täiustamise algoritmist. Katsetulemused on esitatud mitmete suurte tööstuslike skeemide tarvis, mis on realiseeritud XILINX 4000 tehnoloogias. Hierarhial põhinev klasterdamine annab võimaluse kiirendada suurte tööstuslike skeemide tükeldust ja parandada selle kvaliteeti.