8

Circuit clustering and partitioning for system implementations*

Ulrich Weinmann[1] *and Wolfgang Rosenstiel*[2]
[1] *FZI - Computer Science Research Center*
Haid- u. Neustr. 10-14, 76131 Karlsruhe, Germany
[2] *FZI / University of Tübingen, Sand 13, 72076 Tübingen, Germany*
Tel.: (++49) 721 9654 -465, Fax:-409, e-mail:weinmann@fzi.de

Abstract

This paper shows several new perspectives of partitioning circuits into clusters for the implementation into programmable devices. The partitioning algorithm is based on a probabilistic node movement and is able to consider different optimization goals. Special clustering strategies based on the max flow theorem before the partitioning allow the handling of large circuits, a retiming approach following the partitioning improves the overall system performance. Several large benchmark circuits could be implemented into FPGAs with low computational costs.

Keywords

Maximal Flow Clustering, Partitioning, FPGA Synthesis

1 INTRODUCTION

The partitioning of logic structures has become a very important implementation step for the realization of circuits with programmable logic, especially FPGAs (Field Programmable Gate Arrays). These chips are used for both prototyping and the implementation of a low number of produced chips. Partitioning became necessary, because the number of equivalent gates is lower in comparison to mask-programmed gate arrays. Rapid prototyping boards have been developed to emulate large circuits with programmable logic. Here the chip utilisation is less than 40% after partitioning due to the number of external pins on a device. This paper addresses the problem by combining a new partitioning approach with technology mapping in order to achieve better implementation results.

* This research is supported by the Esprit LINK project 6855

G. Saucier et al. (eds.), *Logic and Architecture Synthesis*

Many approaches for partitioning circuits have been proposed to minimize the interconnections between chips. However, an exact solution to this problem can't be found in the sense that no polynomial-time algorithm for it is known. The *clustering partitioning* can be divided into seed selection, unplaced node selection and node placement. This approach is easy to implement, but the quality of the result depends strongly on the selection of the seed nodes (Murgai 1991). Ford and Fulkerson (1962) introduced the dependency between *maximum flow* and minimum cut on a graph. Besides *simulated annealing*, which uses an analogy to the annealing process in physics to avoid local minima, the *Min-Cut two-way partitioning* (Kernighan 1970) is very efficient due to group swapping. These basic partitioning algorithms have been improved in the past, mostly by reducing the calculation complexity to obtain faster partitioning results. Kuznar, Brglez and Kozminski (1993) first showed the necessity of the relationship between partitioning and technology mapping for the Xilinx architectures, using an improved Min-Cut approach.

1.1 Overview

In this paper we try to combine constructive and iterative partitioning approaches in order to meet the user constraints or an overall optimization goal. Figure 1 shows a general overview of the chip partitioning system.

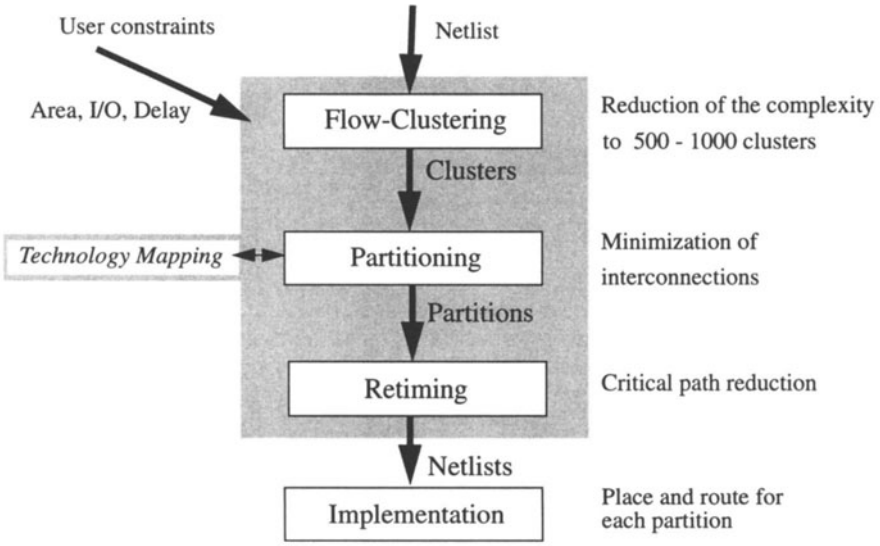

Figure 1 Partitioning overview

Starting from a netlist description, the tool first performs a preclustering of logic structures. The goal of this partitioning step is to reduces the partitioning complexity in order to handle very large circuits. The iterative partitioning step is based on a probabilistic node exchange. Here the user requirements can be considered like partition sizes or pinning. After every optimization run, the technology mapping can be performed to obtain the actual area and delay results for each cluster. The system offers a technology independent mapping tool by Weinmann (1994), which is able to map a logic structure into any SRAM or multiplexer-based architecture.

The timing can additionally be improved by a final retiming step. This operation moves storage elements towards interconnections to reduce the length of critical paths (Leiserson 1983). Afterwards, the mapped circuits are saved and can be implemented by placement and routing tools from the FPGA vendors.

2 CLUSTERING

The partitioning has an exponential complexity with respect to the number of gates and connections. Therefore, larger circuits can only be efficiently partitioned by reducing the number of gates. This leads to the clustering problem, which is based on a grouping of gates (nodes) of a circuit (graph) into clusters. As a result, supernodes or hierarchical netlist descriptions can be partitioned with less effort. One drawback is the reduced search space, which now might not include the optimal result. Avoiding this problem is one goal of the clustering. Furthermore the clustering is able to connect gates, which should not be separated, such as the nodes in a cycle. The following clustering approach is the basis for an efficient partitioning.

2.1 Network flow based clustering

The network flow is a topic in algorithmic graph theory and can be defined as a function for every edge in the circuit. Using this definition, Fulkerson and Ford (1962) first introduced the dependency between maximal flow and minimal cut on a graph. For any network with integer edge capacities, the maximum flow value from the source to the sink is equal the capacity of a minimum cut separating the source and the sink. This theorem has not yet been considered for partitioning, because the cuts often result in very uneven partition sizes. The example in figure 2 shows this problem using a simple graph with the corresponding flows on the edges.

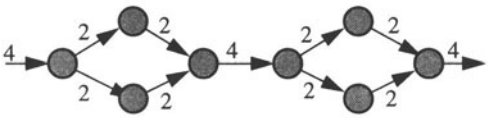

Figure 2 Network Flow

The flow is calculated by counting all paths, which include the edge. The minimum cut can be found on the three edges with maximal flow. The first and last edge can not be used for partitioning, because the partition sizes would be very uneven. Now the goal is, using this theorem, to find edges with a high network flow, which build the basis for the iterative optimization.

The flow based clustering approach uses this information. In the first step the flow of all edges is calculated. Then these edges or their corresponding source nodes are sorted by rising flow. Starting with the node of maximal flow further nodes connected to the inputs are clustered, until their flow is larger than the corresponding output flow (this number can vary and restricts the final sizes of the clusters). The algorithm can be summarized as follows:

```
BEGIN;
FLOW_PRECLUSTERING()
    {
    Calculate Flow on all Nodes
    Sort all Nodes with rising flow in list T
    WHILE (NOT T==∅)
        {
        delete first node q from T
        Node_List(q) = {q};
        CLUSTER(q)
        }
    }

    CLUSTER(v)
        {
        FOR all input nodes v_i of v DO
            IF ( flow(v_i) ≤ flow(v) )
                {
                Node_List(q) = Node_List(q) ∪ {v_i};
                delete node v_i from T
                CLUSTER(v_i)}
                }
        }
END;
```

This can be done until all nodes are assigned to their corresponding clusters. The resulting clusters have minimal connections to their neighbouring clusters. Now the clusters have to be assigned to partitions with the goal of minimal interconnections.

Figure 3 shows a sample circuit including two sequential cycles. In order to calculate the flow on the edges, the cycles have to be cut (usually at the output of storage elements).

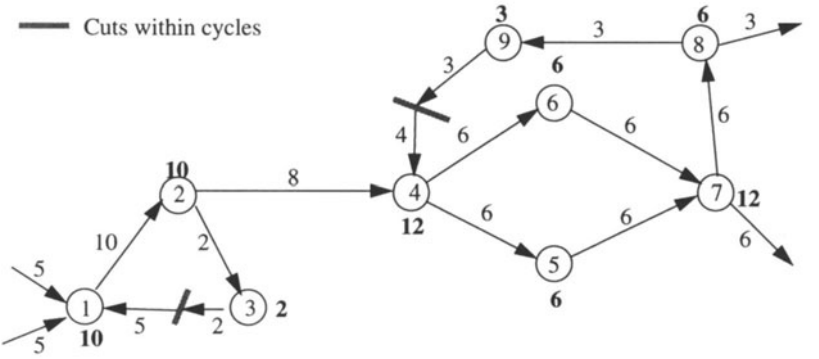

Figure 3 Example for flow-calculation

The flow number on each edge represents the numbers of different paths including the edge. The algorithm now starts with the node of maximum flow (e.g. 7). In direction of the inputs, all nodes are clustered until the flow rises (node 4). The next iteration starts with the max flow of the remaining nodes. Figure 4 shows the resulting clusters.

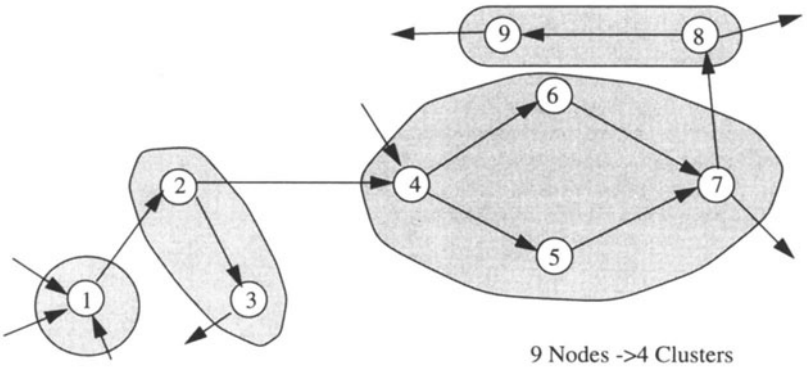

Figure 4 Result after clustering

3 THE PARTITIONING ALGORITHM

This partitioning method is based on a probabilistic cluster exchange between partitions, and includes a dynamic cost function. Given is a graph G = (N,E) with a set of clusters (N) and edges (E), which will be partitioned into k partitions for the implementation into programmable logic (see figure 5).

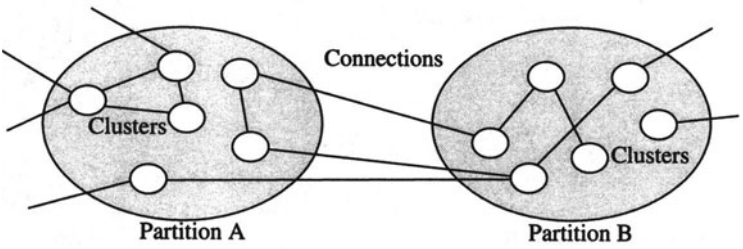

Figure 5 Assignment of clusters to partitions

The partitioning starts with arbitrary partitions, which can be obtained by a simple next-neighbour clustering method. The iterative partitioning allows different weighted optimization factors to be added to the cost function. Using these criteria for each cluster, a *moving probability* p_i of a cluster into partition i can be calculated. To illustrate the strategy, in the following a probability function including two partitioning goals will be described, which then will be extended to a generalized cost function. Here N_i equals the number of clusters in partition i after each partitioning iteration, E_i equals the number of inputs and outputs (I/Os) of partition i and C_{ij} shows the number of connections between the two partitions i and j. These are the results after the partitioning, while uN_i and uE_i are user defined constraints for partition size and input/output count. The probability p_i of a cluster n to move to partition i is equivalent to:

$p_i \sim e_i$ with e_i = Number of interconnections with partition i

and

$$p_i \sim \frac{1}{\delta_i} \quad \text{with } \delta_i = \frac{N_i}{uN_i}$$

That means, the more connections of a cluster to another partition, the higher the probability is for the clusters to move to that partition. Furthermore this probability is inverse proportional to the number of clusters inside a partition. If both criteria have the same weight:

$$p_i \sim e_i \cdot \upsilon_i \text{ with } \upsilon_i = \frac{1}{\delta_i};$$

because of $\sum_{\forall k} p_k = 1$ follows: $p_i = \frac{e_i \cdot v_i}{\sum_{\forall k} e_k \cdot v_k}$ with k = all partitions.

The summation of the probabilities of a cluster to be moved equals 1. By calculating this equation for each cluster, a probability for movement into another partition can be defined. The general probability for cluster movement for several cost functions equals:

$$p_i = \frac{\prod_x a_{ji}^{(\gamma_x)}}{\sum_k \prod_x a_{jk}^{(\gamma_x)}}$$

a_{xi} - Criteria x of cluster i
γ_x - Weight of the criteria x

This function additionally includes weight factors, which allow the user to give different importance to the optimization criteria. If, for example, an exact pinning is important, the weight γ of the I/O criteria would be increased. For very high weight factors the following equation applies, which gives a definite decision for movement:

$$\lim_{\gamma \to \infty} p_i = \frac{a_i^{\gamma}}{\sum_{\forall k} a_k^{\gamma}} = \begin{cases} 1 \text{ for } a_i = \max\{a\} \\ 0 \text{ for } a_i < \max\{a\} \end{cases}$$

At the beginning the weights are predefined, however, during the partitioning a change is possible. As an example, a modification would be necessary, if the allowed number of interconnections between two clusters is exceeded. Cluster movements, which cause additional chip to chip wiring, are within a small range prohibited.

The calculation of a probabilistic movement for clusters within partitions allows an easy integration of any optimization factor, which can dynamically be reconfigured. This is important for the adjustment of the partitioning to user and technology specific data. Figure 6 depicts a simple example for the criteria size and interconnections. For each cluster, k-1 calculations are needed, where k is the number of partitions. The complexity is maximal O(N(k-1)) for N clusters. This corresponds to one iteration for the partitioning. The number of iterations add a linear increase and range depending on the circuit size between 10 and 100. This estimation can be reduced, if only clusters, which are located close to the border of a partition, are considered and therefore have a high probability to be moved.

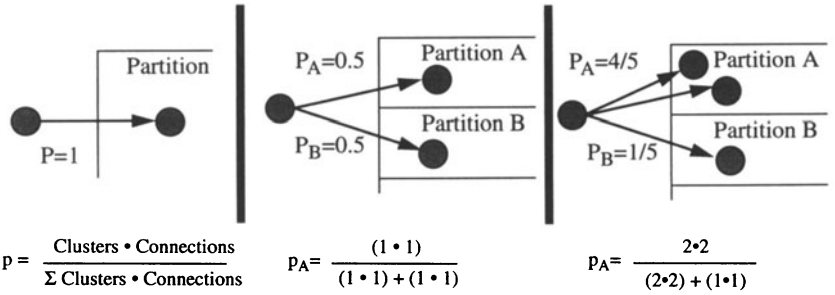

Figure 6 Probability Calculation

3.1 Improved System Performance by Retiming

The retiming operation (Leiserson 1983) can be applied after the partitioning and improves the timing for FPGAs. This technique is based on the movement of storage elements within a circuit and is used to reduce the maximal delay.

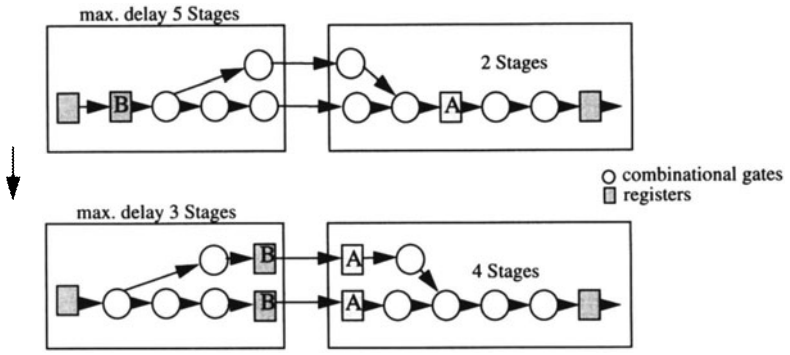

Figure 7 Retiming after the partitioning

The drawback of this operation is a possible increase of the number of registers, if a negative register movement (Weinmann 1993) is applied. In combination with FPGA architectures with predefined storage elements (Xilinx, Actel), the timing can be improved without increasing the overall area consumption on the chip. An evaluation of benchmark circuits (Weinmann 1993) showed, that in the average less than half of the storage elements of the Xilinx architecture are used after mapping. This operation can be applied to every cluster

after the partitioning, however in large designs the overall system performance is most important. Therefore the chip to chip delay has to be taken into account. The lowest delay for partitioned circuits can be achieved by putting storage elements next to the output and input pins of interconnections. This means no additional gate delay to the wiring on the board. Figure 7 depicts an illustration of the basic idea of the retiming operation in combination with the partitioning. The first solution has a maximal delay of five combinational stages in addition to one interconnection. This number can be reduced to three stages by moving register 'A' toward the input of the chip and finally to zero gate delay by moving register 'B'. The only restriction concerns divergent paths with different timing states (retiming bottlenecks), which can be avoided by duplicating gates, described in.

4 RESULTS

Different combinational and sequential MCNC benchmarks (Brglez 1989) were implemented and tested with the partitioning tool. The results are divided into the clustering and partitioning step. Table 1 shows the results after the flow-preclustering for the largest S-Benchmarks.

Table 1 Results after flow-preclustering

Benchmark	*#Gates*	*#Wires*	*Av. # I/O / Gate*	*Flow-Preclustering*			
				#Clusters	*#Wires*	*Av. # I/O / Cluster*	*Time*
s838	422	734	3.5	42	247	11.6	<1 s
s953	424	801	3.8	32	325	20.2	<1 s
s1196	547	1045	3.8	20	330	33.0	<1 s
s1423	805	1534	3.8	126	584	9.2	3 s
s1488	659	1399	4.2	42	471	22.4	2 s
s1494	653	1405	4.3	37	466	25.1	1 s
s5378	2553	4560	3.6	136	710	10.4	21 s
s9234	5825	8427	2.9	293	1395	9.5	43 s
s13207	8620	12503	2.9	817	2407	5.9	77 s
s15850	10369	14839	2.8	798	2718	6.8	87 s
s35932	17793	31275	3.5	1616	7944	9.8	93 s
s38584	20705	35660	3.4	2235	9982	8.8	120 s

Beside the number of gates and wires of the circuit, the average number of inputs/outputs per gate is given. After the preclustering these numbers can be compared with the resulting

clusters and connections. For the larger circuits the number of gates is reduced to about 10% in clusters, however the number of I/Os per cluster only doubled.

Generally partitioning tools are very difficult to compare, because they depend on different parameters and cost functions. In order to achieve a realistic basis for comparison different factors can be calculated to reflect partitioning results under several optimization goals. Very important is the relationship between the number of interconnections and the deviation of the optimal cluster size after the partitioning. Of course, the number of connections can be minimized with a high deviation in the size of the clusters, however, the overall result would be rated less optimal. The average I/O count for each chip for k partitions can be calculated as follows:

$$C_{av} = \frac{1}{k} \sum_{i=1}^{k} \sum_{j=1}^{k} e_{ij}$$

Each edge e_{ij} between the partitions i and j equals a cut between chips. In order to evaluate this number, the relative factor to all edges S in the circuit shows the efficiency of the partitioning.

$$C_{relative} = \frac{C_{av}}{S} \cdot 100\%$$

Together with the number of cuts the use of area in each partition has to be observed. Using an overall circuit size B and a *size* b_i for each partition i, the average deviation a from the average size $\frac{B}{k}$ equals:

$$a = \frac{\sum_{i=1}^{k} \left| \frac{B}{k} - b_i \right|}{B} \cdot 100\%$$

Table 2 depicts the partitioning results for the probabilistic cluster exchange and optimization for the partitioning into two and four partitions. Even the largest partitioned circuits can be implemented in programmable logic, because the number of interconnections stays below the number of available pins of an FPGA package. For the circuit s38584 only 0.44% of the edges in the circuit graph are used for interconnections. The area deviation between the partitions was kept below 25%.

Table 2 Partitioning results for different S-Benchmarks

Bench-mark	*Two Partitions (k=2)*				*Four Partitions (k=4)*		
	Area deviation a	*Av. # I/O* C_{av}	$C_{realtive}$	*Time*	*Area deviation a*	*Av. # I/O* C_{av}	*Time*
s838	0 %	38	5.18 %	23 s	4 %	7	89 s
s953	22 %	55	6.87 %	44 s	12 %	21	32 s
s1196	15 %	39	3.73 %	238 s	23 %	22	82 s
s1423	6 %	34	2.22 %	422 s	19 %	19	136 s
s1488	8 %	55	3.93 %	310 s	12 %	55	272 s
s1494	22 %	49	3.49 %	298 s	16 %	55	356 s
s5378	25 %	147	3.22 %	534 s	13 %	58	752 s
s9234	17 %	104	1.23 %	610 s	25 %	81	923 s
s13207	12 %	121	0.97 %	897 s	24 %	94	1869 s
s15850	25 %	108	0.72 %	745 s	25 %	83	2023 s
s35932	12 %	89	0.28 %	1089 s	23 %	103	1965 s
s38584	23 %	156	0.44 %	1278 s	21 %	112	2403 s

5 CONCLUSION

This paper proposes a new partitioning method divided into several optimization steps. Using maximal flow calculations, a new clustering algorithm was presented. This information could be used for a probabilistic cost function to optimize area and I/O count. Additional optimization steps for delay reduction improve the performance of the partitioned system.

6 REFERENCES

Brglez, F., Brayan, F. and Krzysztof, D. (1989) Combinational Profiles of Sequential Benchmark Circuits. *Technical Report TR89,* Microelectronics Center of North Carolina.

Ford, L.R. and Fulkerson, D.R. (1962) Flows in Networks. *Princton University Press.*

Kernighan, K.H. and Lin, S. (1970). An Efficient Heuristic Procedure for Partitioning Graphs. *Bell System Technical Journal*, vol. 49, no. 2.

Kuznar, R., Brglez, F. and Kozminski, K. (1993) Cost Minimization of Partitions into Multiple Devices. *Proceedings of DAC 93*, 315-20.

Leiserson, C. E., Rose, F. M. and Saxe, J. B. (1993) Optimizing synchronous circuitry by retiming. *Proc. Third Caltech Conf. on VLSI.*

Murgai, R., Brayton, R. and Sangiovanni-Vincentelli, A. (1991) On Clustering for Minimum Delay/Area. *Proceedings of DAC 91*, 6-10.

Weinmann, U. (1993) FPGA Partitioning under Timing Constraints. *Int. Workshop on Field Programmable Logic and Applications*, Oxford.

Weinmann, U. and Rosenstiel, W. (1993) Technology Mapping for Sequential Circuits based on Retiming Techniques. *Proceedings of the EURO-DAC 93*, 318-23.

Weinmann, U. and Rosenstiel, W. (1994), Module Independent Mapping into TLU-FPGAs. *2nd ACM/SIGDA FPGA Workshop*, Berkeley.