

Partitioning for Multicomponent Synthesis from VHDL Specifications

Nand Kumar, Ram Vemuri, and Ranga Vemuri
Laboratory for Digital Design Environments
Department of ECE, Mail Location 30
University of Cincinnati, Cincinnati, OH 45221
e-mail: nand@thor.ece.uc.edu

Abstract: This paper presents a technique used for constraint-driven partitioning of VHDL designs for multicomponent synthesis. Hard constraints (area, clock speed, heat dissipation, and number of structures) on the entire design and on the individual structures (area, clock speed, pin count, and heat dissipation) are satisfied during the process of partitioning. The technique uses a partitioning engine that provides the user with a mix of interactive and numerous automatic partitioning approaches. PDL is used for performance specification and measurement.

1 Introduction

Current VLSI designs, due to their complexity and size, are often too large to fit on a single chip. With the increasing complexity of digital designs and the continuing revolution in interconnect and packaging technology (multi-chip modules), computer-aided partitioning techniques must be used to realize efficient multi-chip implementations that utilize the available technology.

Gupta and De Micheli [4] use the Kernighan-Lin and simulated annealing techniques for partitioning functional models while satisfying area and timing constraints. Pin-sharing or area/delay characteristic of registers, multiplexers, controllers, or wiring are not considered. Shih, Kuh, and Tsay [16] use a clustering step to satisfy timing constraints before using the Kernighan-Lin algorithm to partition functional blocks. McFarland uses a hierarchical clustering technique in partitioning behavioral hardware descriptions [10]. Design constraints are not considered. Lagnese and Thomas use a multistage clustering technique to choose partitions [8]. The approach shows significant area reductions, but does not consider design constraints.

Kucukcakar and Parker [7] describe CHOP, a framework for interactive partitioning, in which the designer creates and modifies partitions and CHOP evaluates the validity of each partition by searching for possible implementations through predictions. SPARTA evaluates an RTL design with a spreadsheet like approach and checks for the violation of area, power, and pin count constraints [12]. The SLIP framework [1] consists of: a collection of routines that manipulate the design data, an attribute mechanism, and models built from the attributes. SLIP is used to partition hierarchical designs composed of structural entities. The framework supports implementations of simulated annealing, min-cut, and clustering techniques.

Vahid and Gajski [17] describe partitioning at the algorithmic level. Clustering and Kernighan-Lin algorithms are used in partitioning. A preliminary synthesis of behavioral objects in the design is performed prior to partitioning. This can lead to inaccurate estimates of performance measures.

We present a constraint-driven partitioning technique that partitions digital systems, specified in VHDL, into multiple structures to realize an efficient multicomponent implementation. The design can be specified at multiple levels of abstraction using VHDL. The technique uses a partitioning engine that supports design specification at the behavioral and register transfer levels of abstraction. When the input is behavioral VHDL, the output consists of multiple behavioral VHDL specifications that should be synthesized separately using a high level synthesis system [14]. When the input is register level VHDL, the output consists of multiple register level VHDL designs. Hierarchical clustering, genetic evolution, direct partitioning, and interactive partitioning algorithms are currently supported by the partitioning engine. Other algorithms can be added as needed. The partitioning engine provides routines for design management and attribute evaluation in addition to being the interface between the user and the various partitioning algorithms.

Section 2 describes the formulation of the partitioning problem. Section 3 outlines the partitioning technique using the partitioning engine and briefly describes some of the algorithms used. The PDL (Performance Description Language) evaluators and some attribute evaluation techniques are described in Section 4. Some implementation details and preliminary results are presented in Section 5.

2 Formulation of the Partitioning Problem

The design specification needs to be partitioned into multiple structures for synthesis. In the case of a behavioral specification, the design is partitioned into multiple behavioral segments which are synthesized separately using a synthesis system [14]. In the case of a register transfer level specification, the design is partitioned into multiple register level structures. The designs are composed of register level modules from the design library. The designs produced are optimal with respect to certain predefined performance metrics and constraints. The performance measures considered are: area, clock speed, heat dissipation, and number of partitions or structures for the entire design and area, clock speed, pin count, and heat dissipation for the individual partitions or structures. PDL [9] annotations, to the input VHDL, are used to specify the constraints. Performance evaluation routines, for each performance attribute, are written in PDL and made available to the partitioning engine. These evaluators can be written to any desired precision.

The input behavioral VHDL specification, in the form of a data-flow graph (the VHDL Intermediate Form - VIF [13]) (alternately, in the form of a composition of register level components from the design library), is to be partitioned. The result of partitioning is a set of VIF segments (alternately, a set of register level structures). Each VIF segment then goes through the design process - detailed scheduling, allocation, and binding [14]. In the case of a behavioral specification, an initial partition is identified along the lines of VHDL processes, blocks, components, subprograms, and concurrent signal assignments. Component instantiations indicate predefined structures that must be preserved during the synthesis process.

The partitioning problem is a combinatorial problem; even its restricted versions are known to be NP-hard [11]. The problem is formulated as a constraint satisfaction problem. Consider the initial partition to consist of VIF segments (alternately, register level components), $P = \{p_1, p_2, \dots, p_n\}$, called *process bodies* or *processes* (for short). Processes are grouped together into several disjoint subsets, $G = \{g_1, g_2, \dots, g_l\}$ for some $1 \leq l \leq n$, called *groups*. These groups in turn are partitioned into disjoint subsets, $S = \{s_1, s_2, \dots, s_m\}$ for some $1 \leq m \leq l$, called *segments*. In the case of behavioral specifications each s_i is mapped onto a single register level structure either as a direct result of partitioning or after synthesis of the individual behavioral segments (resulting in a data path + control graph). From this partition other partitions are created by iteratively merging two or more segments to form a larger segment and/or by migrating one or more subsegments (groups) from one segment to another. Subsets of the set S of segments are then bound to specific structures

to create a set of structures $B = \{b_1, b_2, \dots, b_k\}$ for some $1 \leq k \leq N$ where N is the constraint on the number of individual structures in the design. Figure 1 shows an example design and its hierarchy. The partition B is subject to the following set of constraints:

1. *Individual Structure Constraints:* The performance of individual structures is defined by:

$$\sum_{s_i \in b_k} a_j(s_i) \leq c_j \quad (1)$$

where, a_j denotes the performance attributes of the individual segments in the partition, b_k denotes the individual structures, s_i denotes the segments in each structure, and c_j denotes the user specified constraint on the attribute. Each a_j is computed by a PDL program which takes the design segment as input. Example a_j s include area, heat, clock-period, number of pins.

2. *Overall Design Constraints:* The overall design performance is defined by:

$$A_j(B) \leq C_j(\forall j) \quad (2)$$

where, A_j denotes the performance attributes of the entire design and C_j denotes the user specified constraint on the attribute - j is the total number of constraints on the overall design. A_j is computed by a PDL program that takes the set of structures and their configuration profile as input. Example A_j s include area, heat dissipation, and number of structures in B .

The PDL program is chosen, from a library of evaluators, based on the technology in which the structure (and its constituent segments) are to be implemented.

3 The Partitioning Technique

The partitioning problem is formulated as a constraint satisfaction problem in Section 2. Many techniques solve restricted versions of this problem. Among the techniques are direct partitioning techniques, group migration, and metric allocation techniques [11]. Combinatorial optimization techniques such as simulated annealing [6] and stochastic evolution [15] have been applied to partitioning. Genetic evolution [3] is a promising new technique for partitioning, that works well in the presence of multiple constraints [15]. Our partitioning approach makes use of some of the existing partitioning approaches in addition to investigating new techniques such as genetic algorithms.

The partitioning technique is based on giving the user complete control over the selection of the algorithm to be used for partitioning. To facilitate the integration of various algorithms and user interaction, a *partitioning engine* is used. Section 3.1 discusses the partitioning engine and Section 3.2 briefly outlines some of the partitioning algorithms.

3.1 The Partitioning Engine

The *partitioning engine* (Figure 2) is a collection of routines that supports the use of multiple partitioning techniques, manages the design (incorporating changes as indicated by the partitioning algorithms), evaluates the attributes of the design, and provides the interface between the user and the partitioning algorithms.

In the case of a behavioral specification, objects identified along the lines of VHDL processes, blocks, components, and subprograms are scheduled (subject to constraints) and performance estimation is

1, 2, 3, 4, 5, 6, 7, 8: process

g1, g2: group

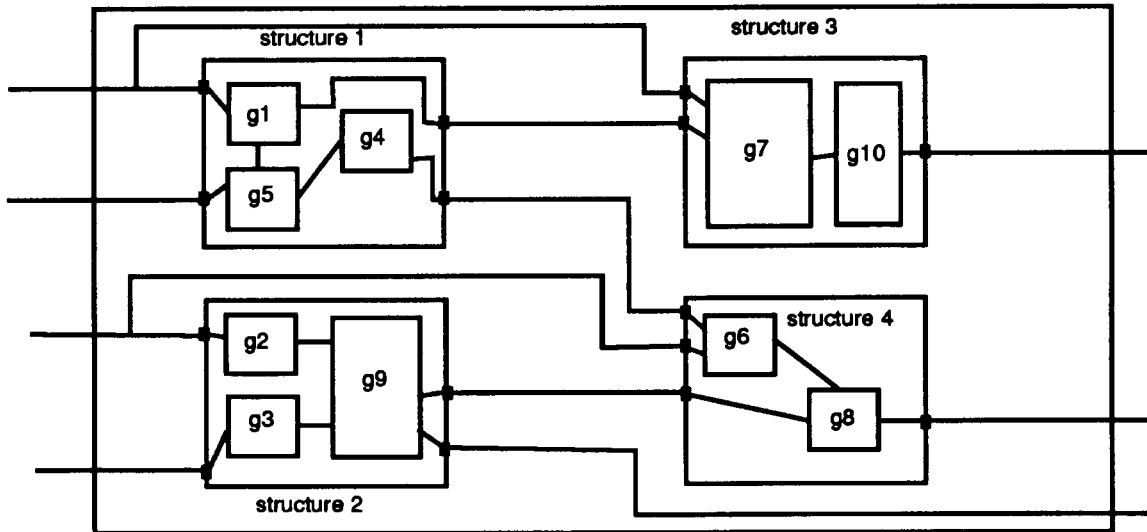
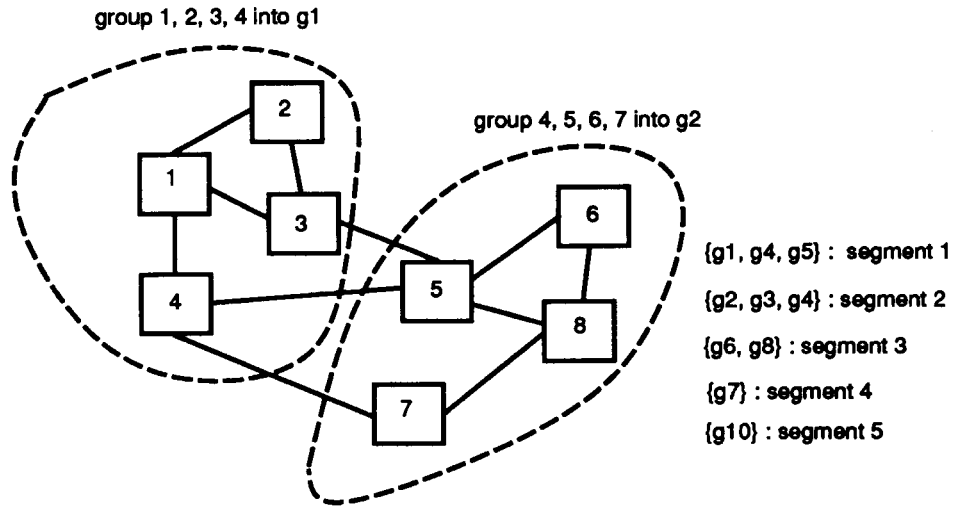


Figure 1: An example design and its hierarchy

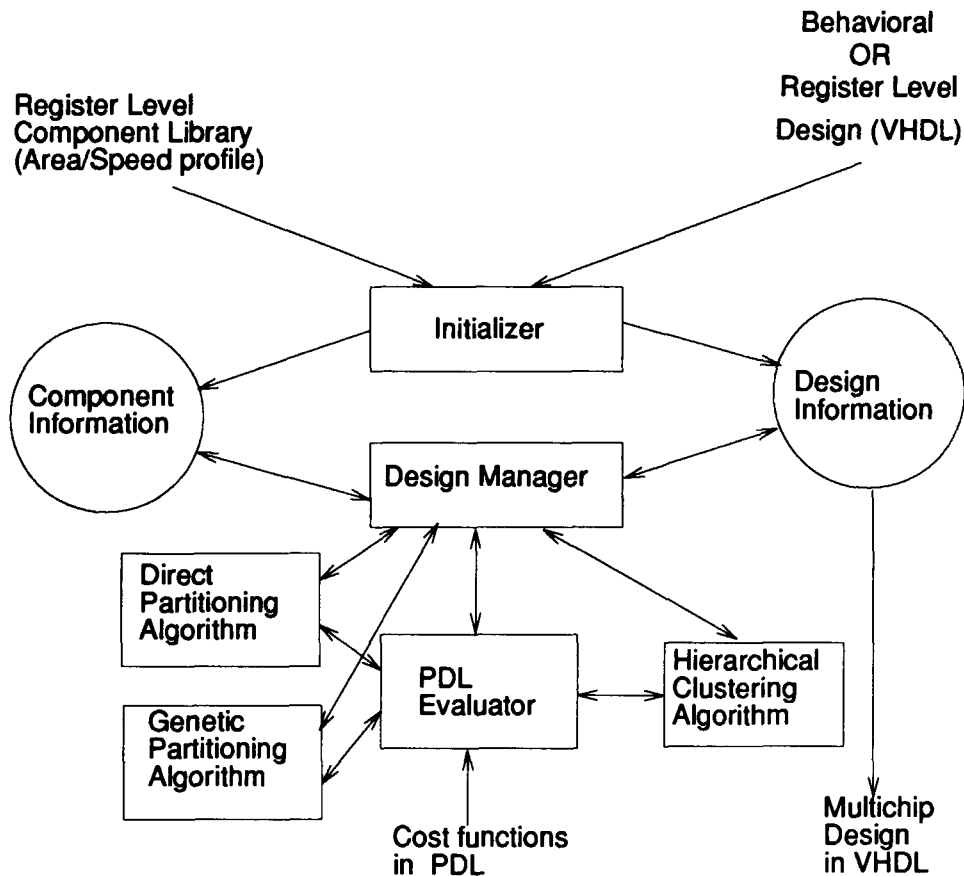


Figure 2: The Partitioning Engine

carried out to obtain the attribute values prior to partitioning¹. As the design evolves through the partitioning process, the attributes of the design are constantly updated through re-application of the scheduling and performance estimation steps. In the case of a register level design, the attribute values of the register level components are available from the design library.

The engine has access to a library of partitioning techniques. Hierarchical clustering, genetic evolution, direct partitioning algorithms, and interactive partitioning algorithms are currently supported by the partitioning engine. Other partitioning algorithms can be added as needed. The partitioning engine has access to a library of PDL evaluators (Section 4).

3.2 Partitioning Algorithms

The solution techniques to the partitioning problem can be broadly classified into two categories: direct partitioning techniques that use heuristics and randomized search techniques. The direct techniques include group migration, metric allocation, and clustering algorithms [11, 5, 10, 8]. The randomized search techniques consist of methods such as stochastic evolution, simulated annealing and genetic algorithms [6, 3, 15]. In this section we present the algorithms that are currently supported by the partitioning engine.

¹Scheduling and performance estimation are integral parts of the high level synthesis process [14].

Hierarchical Clustering: The clustering techniques developed until now do not consider design constraints [10, 8]. Our clustering technique uses a constraint-driven hierarchical clustering approach to improve the quality of the partitions produced. The clustering technique groups *close* design segments together. A *closeness* measure is computed for all pairs of the design segments based on the communication between them. Clusters of design segments are composed using the closeness measure. The clusters thus formed are subject to further clustering at the next hierarchical level. At each level of clustering the performance attributes of the partitions are computed and verified against the constraints. The feasibility of the partitions thus created is determined by performance prediction using the PDL evaluators. This ensures that the design constraints are met by the partitions and thus the final design.

Genetic Algorithms: Randomized search algorithms which intermittently allow inferior designs in order to reach a final optimal design are known to be effective in solving combinatorial problems. Genetic algorithms [3] are one such class of algorithms. A genetic algorithm is a randomized parallel search method modeled on natural selection and genetics[3]. In contrast to more standard search algorithms, GAs base their progress on the performance of a population of candidate solutions, rather than on a single candidate solution.

Our genetic partitioning algorithm, shown in Figure 3, initially creates a number of random partitions. The *fitness* of a partition, a measure of its goodness, is defined as,

$$\frac{1}{1+\sum CD}$$

where CD is the net deviation of any constraint of each partition. Fitness is a value between 0.0 and 1.0. The algorithm uses the following operators in order to generate the next generation of partitions:

- *Selection:* A randomly selected partition which is highly fit (fitness value > 0.8) can be moved into the next generation. Twenty percent of the partitions in the new generation are created using selection.
- *Crossover:* Two highly fit partitions P_1, P_2 in the current population are randomly selected. The largest structure (chip) C (in terms of the value of the *area* attribute) in P_1 is copied into P_2 . Design segments in C which are currently assigned to other structures (chips) in P_2 are deleted from those structures (chips). Both P_1 and P_2 are moved into the next generation of partitions. Eighty percent of the partitions in the new generation are created using crossover.
- *Mutation:* Mutation involves randomly selecting a partition and moving a design segment from some randomly selected structure (chip) in the partition to another randomly selected structure (chip). Mutation is applied to twenty percent of the partitions in the new generation.

The genetic partitioning algorithm terminates when a termination criterion is satisfied. The criterion usually is that a partition with fitness 1.0 be found. In addition, to find optimal partitions (not just constraint satisfying partitions), other criteria such as the computation time, number of generations to be searched or a lower limit on a measure of global optimality such as the total number of interconnection wires or the total number of chips are also specified. The genetic partitioning algorithm is quite slow but can find better quality partitions than the other algorithms.

Direct Partitioning Using Heuristics: All the direct partitioning techniques [5, 2] use deterministic heuristics to solve the partitioning problem. The Kernighan and Lin algorithm starts with

```

genetic partitioning algorithm
1  N: population size (number of partitions in a generation) := 100
2  S: percent of new generation produced by selection := 20
3  C: percent of new generation produced by crossover := 80
4  M: percentage of partitions mutated := 20
5  begin
6      create a random set of N partitions.
7      evaluate the fitness of each partition.
8      while (stopping criteria not satisfied) do
9          begin
10             Create S percent of new population of partitions by selection.
11             Create C percent of new population of partitions by crossover.
12             Replace the current generation by new generation of partitions.
13             Mutate M percent of the current partitions.
14             Evaluate the fitness of each partition.
15             Save the partition with the best fitness.
16         end
17     end

```

Figure 3: Genetic Partitioning Algorithm

a random initial partition and then applies pairwise swapping (of subsets) between the two partitions with constraints on the subset sizes [5]. Fiduccia and Mattheyses improved the Kernighan-Lin algorithm by reducing time complexity $O(P)$ with respect to the number of pins, P , in the circuit [2]. We provide the partitioning engine with an adaptation of the Fiduccia-Mattheyses algorithm.

Interactive Partitioning: The user does the partitioning and uses the engine for design management and performance evaluation to check if the constraints are satisfied.

4 PDL Evaluators

The library of PDL evaluators consists of parameterized evaluators written for each performance attribute. A PDL program [9] takes a design segment as its input and returns the value of a particular performance attribute (such as area, heat, speed etc). PDL facilitates writing such evaluators precisely and concisely and to arbitrary precision as desired by the user. New evaluators can be added to the library for new performance attributes or to improve the precision of the existing evaluators. Since the partitioning engine and the algorithms the engine uses are independent of the constraint and attribute evaluation mechanisms, the above discussed modifications to the performance estimators is possible *without* modifying the partitioning approach or algorithms. PDL evaluators are used to check if constraints are met during the process of partitioning and also in the verification of performance constraints after the design process is completed.

Example	Beh. VHDL Number of Lines	No. of Components	No. of Nets
Fifo	65	69	244
Find	65	90	338
Move Machine	105	49	302
TLC	45	49	143
Viper	350	156	970

Table 1: Design Data for the Examples

Example	No. of Chips	Number of Pins/Area(sq mm)			Execution Time (sec)
		Chip1	Chip2	Chip3	
Fifo	3	73/4.7	70/4.7	70/1.5	57
Find	2	70/4.3	80/7.8	-	94
Move M/c	2	88/7.4	84/9.9	-	29
TLC	2	31/2.9	40/2.4	-	11
Viper	2	219/13	199/19.7	-	960

Table 2: Results using the Fiduccia-Mattheyses Algorithm

5 Implementation Details and Preliminary Results

The partitioning engine currently consists of 6000 lines of C++, the genetic algorithm consists of 1000 lines of C, the Fiduccia-Mattheyses algorithm consists of 1000 lines of C++, and the hierarchical clustering algorithm is implemented in 150 lines of Prolog.

Experimental results for some VHDL descriptions are provided. These examples range from a simple traffic light controller to the Viper microprocessor. More details about the examples can be found in [18]. The partitioning was carried out on register level VHDL descriptions produced after high level synthesis was carried out on the behavioral VHDL using a high level synthesis system (DSS [14]). Table 1 gives some design data for the examples. Tables 2, 3, and 4 present the results for the Fiduccia-Mattheyses (F-M) algorithm, genetic partitioning algorithm, and for the hierarchical clustering algorithm respectively. Comparing the results obtained using the three techniques, the F-M and the clustering algorithms run faster than the genetic algorithm, but the genetic algorithm consistently produces better results. The F-M algorithm performs better than the hierarchical clustering algorithm.

6 Acknowledgments

This research is sponsored in part by the Solid State Electronics Directorate, Wright Laboratory, WL/ELED, Air Force Systems Command, United States Air Force under contract number F33615-91-C-1811.

Example	No. of Chips	Number of Pins/Area(sq mm)			Execution Time (min)
		Chip1	Chip2	Chip3	
Fifo	3	74/5.9	66/2.9	55/2.2	33
Find	2	66/3.8	75/8.2	-	69
Move Machine	2	57/7.8	60/9.5	-	3
TLC	2	25/0.5	30/4.8	-	14
Viper	2	99/29.9	79/2.8	-	410

Table 3: Results using the Genetic Algorithm

Example	Number of Pins/Area(sq mm)							Exec. Time
	Chip1	Chip2	Chip3	Chip4	Chip5	Chip 6	Chip7	
Fifo	115/6.0	100/5.0	-	-	-	-	-	3 mins.
Find	83/3.9	93/2.1	71/1.6	90/3.8	77/0.7	-	-	2 mins.
Mv Mc	100/6.4	123/10.9	-	-	-	-	-	3 mins.
TLC	56/4.0	49/1.3	-	-	-	-	-	2 mins.
Viper	180/2.73	181/7.8	258/3.02	265/2.85	308/2.9	243/2.2	300/13.4	30 mins.

Table 4: Results using the Clustering Algorithm

References

- [1] M. Beardslee, C. Kring, R. Murgai, H. Savoj, R.K. Brayton, and A.R. Newton, "SLIP: A Software Environment for System Level Interactive Partitioning", Proceedings of ICCAD, pp. 280-283, 1989.
- [2] C.M. Fiduccia and R.M. Mattheyses, "A Linear-Time Heuristic for Improving Network Partitions", Proceedings of the 19th Design Automation Conference, pp. 175-181, June 1982.
- [3] D. Goldberg, "Genetic Algorithms in Search, Optimization, and Machine Learning", Reading, MA, Addison-Wesley, 1989.
- [4] R. Gupta and G. De Micheli, "Partitioning of Functional Models of Synchronous Digital Systems", Proceedings of ICCAD, Santa Clara, pp. 216-219, November 1990.
- [5] B.W. Kernighan and S. Lin, "An Efficient Heuristic Procedure for Partitioning Graphs", The Bell System Technical Journal, pp. 291-307, February 1970.
- [6] S. Kirkpatrick, C.D. Gelatt, Jr, and M.P. Vecchi, "Optimization by Simulated Annealing", Science, Vol. 220, pp. 671-680, May 1983.
- [7] K. Kucukcakar and A.C. Parker, "CHOP: A Constraint-Driven System-Level Partitioner", Proceedings of the 28th ACM/IEEE Design Automation Conference, pp. 514-519, June 1991.
- [8] E.D. Lagnese and D.E. Thomas, "Architectural Partitioning for System Level Design", Proceedings of the 26th ACM/IEEE Design Automation Conference, pp. 62-67, June 1989.
- [9] R. Mandayam and R. Vemuri, "An Executable Notation for Performance Specification & Evaluation", TM-DDE-92-10, University of Cincinnati, 15 pp., February 1992.
- [10] M.C. McFarland, "Computer-Aided Partitioning of Behavioral Hardware Descriptions", Proceedings of the 20th Design Automation Conference, pp. 472-478, June 1983.

- [11] B. Preas and M. Lorenzetti (eds), "Physical Design Automation for VLSI Systems", Benjamin Cummings Pub., 1988.
- [12] M.L. Resnick, "SPARTA: A System Partitioning Aid", *IEEE Transactions on Computer-Aided Design*, Vol. 5, No. 4, pp. 490-498, October 1986.
- [13] J. Roy, "The VSS Intermediate Format (VIF)", pp. 11, Design Memo, Lab. for Digital Design Environments, University of Cincinnati, February 1991.
- [14] J. Roy, N. Kumar, R. Dutta, and R. Vemuri, "DSS: A Distributed High-Level Synthesis System", *IEEE Design & Test of Computers*, pp 18-32, June 1992.
- [15] Y. Saab and V. Rao, "An Evolution-Based Approach to Partitioning ASIC Systems", Proceedings of the 26th ACM/IEEE Design Automation Conference, pp. 767-770, June 1989.
- [16] M. Shih, E.S. Kuh, and R-S. Tsay, "Performance-Driven System Partitioning on Multi-Chip Modules," Proceedings of the 29th ACM/IEEE Design Automation Conference, pp. 53-56, June 1992.
- [17] F. Vahid and D. Gajski, "Specification Partitioning for System Design," Proceedings of the 29th ACM/IEEE Design Automation Conference, pp. 219-224, June 1992.
- [18] R. Vemuri, J. Roy, P. Mamtora, and N. Kumar, "Benchmarks for High Level Synthesis," Technical Report TM-DDE-91-11, Lab. for Digital Design Environments, University of Cincinnati, June 1991, Revised November 1991.