SYSTEM-LEVEL DESIGN GUIDANCE USING ALGORITHM PROPERTIES

Lisa Guerra, Miodrag Potkonjak ‡, Jan Rabaey

Department of Electrical Engineering and Computer Sciences
University of California
Berkeley, California, USA

‡C&C Research Laboratories, NEC, Princeton, New Jersey, USA

ABSTRACT - This paper introduces an approach which provides quantitative information used to aid in making system-level design decisions such as algorithmic or architectural selection. The method is based on the idea of identifying and using the size and structural properties of algorithms, which affect design performance. These properties provide insight in the matching of an algorithm and a particular implementation platform and a link between algorithms and architectures. An in-depth study of three properties - concurrency, temporality, and regularity - is presented in the context of ASIC area estimation. The underlying intuition behind them and quantitative definitions are given. In addition, illustrations of their utility as estimators of implementation performance are shown using both examples and empirical studies.

1. INTRODUCTION

Common system-level design decisions include the choice of algorithms for a given functional specification, the partitioning and optimization of those algorithms, and the selection of architectures. Case studies indicate that these choices often improve performance by an order of magnitude or more [Pot93, Lid94]. When doing system-level design, it is thus important to be able to analyze the possibilities and trade-offs of high level decisions, before investing a lot of effort in exploration at lower levels. Because of the size and complexity of the design space, however, the designer often cannot adequately explore many of these possibilities.

Techniques which aid in more efficiently and quantitatively exploring the system-level design space will become increasingly important and necessary. This paper proposes an approach for providing such guidance in the system-level design process. The underlying idea is that the performance metrics of a final design can be traced to properties of the algorithm itself. The basis of this work thus involves identifying, measuring, and collectively using the size and structural properties of algorithms which affect design performance on different architectural platforms. First, an overview of the approach is given, followed by a case study demonstrating the use of properties for ASIC area estimation.

2. RELATED RESEARCH

Several previous works deal with analyzing properties on a set of applications to provide design guidance for general purpose architectures. For example, Amdhal's law [Amd64] indicates the amount of speedup attainable on an algorithm by using a parallel architecture. Locality and the 90/10 rule of thumb, convey the qualitative

0-7803-2123-5/94 $4.00

property of typical programs that 90% of execution time is typically spent on 10% of the instructions [Knu71]. In the VLSI DSP domain, the qualitative observation that many signal processing algorithms are regular, has motivated the development of systolic and wavefront arrays [Kun88]. The use of algorithm properties has also been proposed in the parallel computing research. Jamieson [Jam87] proposed a characterization for parallel algorithms to be mapped onto parallel architectures. In addition, Papaefstathiou [Pap93], described a framework for parallel software performance prediction based on algorithm characterizations. In hardware-software codesign research, use of several metrics for guiding partitioning has also been proposed (e.g. [Tho93]).

In this work, a comprehensive and organized treatment of structural properties is presented. The properties form the basis of a unified methodology for system-level design guidance. Previous works have dealt primarily with homogeneous resources (e.g. ALUs), while here a heterogeneous mix of resources is supported. Also, the approach is completely based on using quantitative definitions of all the properties. Quantification of the temporal density, concurrency, and regularity measures are presented in this paper. An overview of the approach follows.

3. DESIGN GUIDANCE ENVIRONMENT

The goal of the proposed environment is to provide quantitative *design guidance* (Figure 1). *Design guidance* involves aiding in tasks at the system-level such as algorithm and architecture selection. For these tasks, performance predictions (e.g., cost, throughput) are used to compare design solutions as opposed to actually performing the time-consuming hardware and software synthesis. Design guidance also involves providing aid to hardware synthesis tasks (e.g., transformations, partitioning). Other types of feedback include suggestions for architecture instruction sets that are better suited for the algorithm, and identification of architecture subcomponents that are in high demand but short supply (or vice versa). Fulfillment of these goals is the topic of ongoing research in our research group.

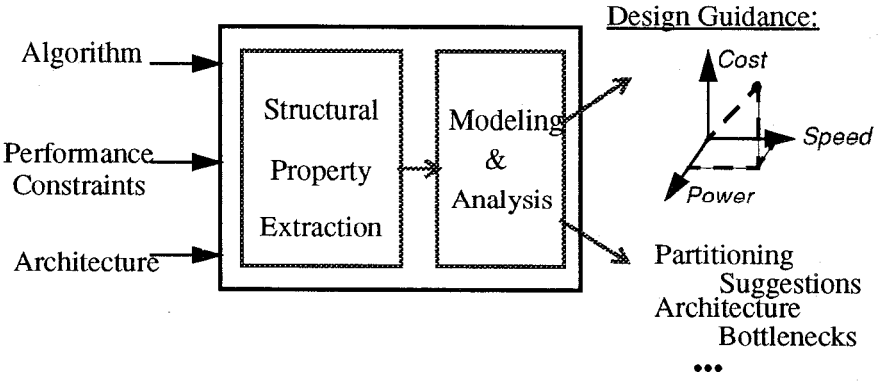


Figure 1: Design Guidance using algorithm properties

Targeted applications are real-time DSP and numerically intensive applications, including a variety of filters, transforms, elementary functions, and speech and

image subsystems. In terms of architectures, application specific datapath-intensive architectures have been of primary interest so far, but the proposed methodology and properties is being used for other DSP architectures as well (e.g. programmable DSPs).

As mentioned in Section 1, the basis of the approach is that performance metrics of a final design can be traced to properties of the algorithm itself. For example, the *number of operations* is directly related to the implementation speed on single datapath uni-processors. The suitability of an algorithm for a targeted application, however, is a function of not only size measures but also other structural qualities. This

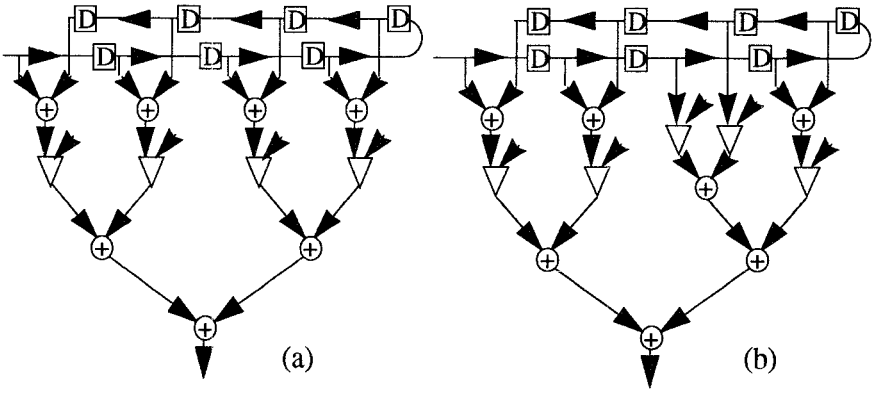


Figure 2: Size versus Concurrency Structural Property

claim is well illustrated using the symmetric direct-form FIR filter of Figure 2a. Figure 2b shows the filter after distributivity is applied to one of the addition-multiplication pairs. It is assumed that an ASIC implementation is being used, where each operation takes 1 control step and the required throughput is 4 control steps. In both graphs, the critical path is 4, and all operations lie on the critical path. Note that the original filter structure has 7 additions and 4 multiplications, while the transformed one has the same number of additions but an additional multiply operation. With hardware sharing, the implementation of Figure 2a requires 4 adders and 4 multipliers, while that of Figure 2b requires just 3 adders and 3 multipliers. Although the transformed filter has a greater number of operations, it has a structure which can be implemented with fewer execution units. Use of the *concurrency* measure (Section 4.1) could aid in selecting between the two structures, or could be used to guide the transformation from the first to the second.

We propose the following set of properties for algorithm characterization. While the set is not claimed to be exhaustive, it has provided a good starting point from which to build our framework:

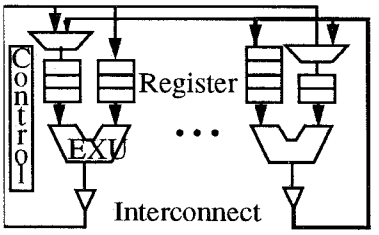
- *Size* measures includes quantities such as the number of nodes, the bitwidth, the number of i/o operations, and the number of memory accesses.

- *Timing freedom and structural freedom* are quantified through measures such as the ratio of sample rate to the critical path, the number and types of nodes on the ε-critical network, and various statistics of the scheduling slack.
- *Uniformity* captures the degree to which operations and interconnect accesses are evenly distributed in time over the course of the computation.
- *Concurrency* measures the number of operations and interconnect accesses that can be executed concurrently.
- *Temporality* captures information about the lifetimes of variables in the computation. A computation is considered to be *temporally local* if the expected lifetimes of the variables are short. It is *temporally dense* if the measured maximum expected number of variables alive at any time is large.
- *Spatial locality* characterizes the degree to which the algorithm has natural clusters of computation, within which significant amounts of computation can be done independently.
- *Regularity* captures the degree to which common patterns appear.
- *Cyclic properties* involve measuring various characteristics of cycles such as the iteration bound and percentage of nodes in cycles.
- *Control flow properties* involve determining the structure and interaction between loops, and properties related to multi-threaded implementations.

The properties themselves are independent of any particular implementation platform, and characterize only the computation. The architecture however, determines which properties are relevant for a given design guidance task.

4. CASE STUDY: Early Estimation of Custom ASIC Area

The properties and the general methodology for using them within a design guidance framework is best introduced through examples. This section thus looks



Component	Relevant Properties
EXU	operator concurrency, size
Register	temporal density, size
Interconnect	operator & interconnect concurrency, regularity, size

Figure 3: Custom ASIC implementation platform and relevant properties

at the properties in the context of one design guidance task, early estimation of custom ASIC implementation area. The hardware platform of interest is shown in Figure 3 and consists of hardware-shared execution units, registers, and interconnect. A finite state machine based controller is used to drive the datapaths.

The following sub-sections consider the estimation of each of the datapath components, and the properties that are relevant for their estimation. In particular, the

relations between components and properties that have been established are shown in Figure 3.

4.1 Operator Concurrency

The execution unit area depends on both the number of units of each type, and the area of each unit. The latter is a function of the required operation word length, which is a *size* property. The number of units is related to among other things, *operator concurrency*, which will be the focus of this section.

The operator concurrency measures the expected number of operations that must be simultaneously executed.The first step in its calculation involves constructing a distribution graph, which quantifies an expected number of operations to be executed in each clock cycle. Paulin first proposed and used information from such graphs in force directed scheduling [Pau89]. As the exact times in which each operation will be executed is not known until after scheduling, it is necessary to determine the probability that the operation will be executed in a particular time step. One solution [Pau89] is to assume that it is equally likely that an operation, x, be executed in any of the time slots between its as-soon-as-possible ($ASAP_x$) and as-late-as-possible ($ALAP_x$) times. In this case, for $ASAP_i \leq t < ALAP_i$,

the contribution $\frac{1}{ALAP_i - ASAP_i + 1}$ is made by an operation i

to each time slot $t \leq k < t + Duration_i$. The values of the distribution graph are the accumulation of the contributions from all operations of the specified type.

From the distribution graphs, several operation concurrency metrics can be measured. One measure is the maximum height of the distribution graph. This is an approximation of the maximum number of operations that need to be executed simultaneously, which therefore relates to an approximate number of required resources. Another measure is maximumheight' = Max(1,maximumheight). This is

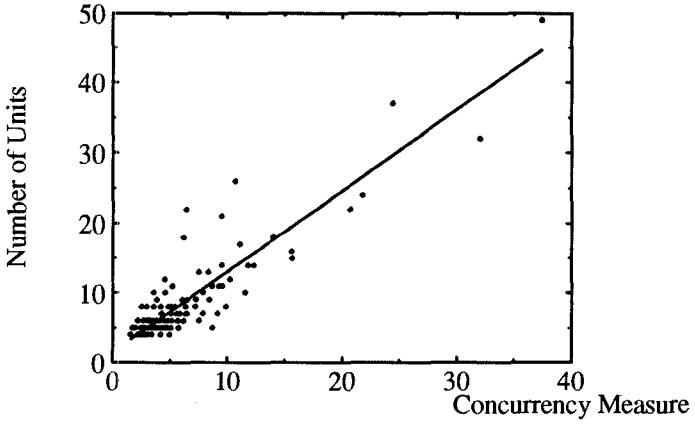


Figure 4: Actual Number of Units versus Maximum Height

used since the number of units must always be at least one. Also of interest is the variance of the distribution graph, a *uniformity* measure, which while not relevant

for execution units, indicates the potential of transformations which target alterations of ASAP and ALAP times (e.g. retiming).

An empirical study was conducted to detect the trends between actual numbers of units in implementation, and the measures. For each of 140 examples, the measures were computed, then the number of units was determined using the HYPER [Rab91] synthesis system. The scatter plot of Figure 4 shows the strong correlation that exists between the max height and the actual number of units. For each design, the sums of the maximum heights for each hardware class was used. The linear correlations of the total number of units to the sums of maximum height, and sums of maximum height' were 0.92 and 0.93, respectively. This demonstrates that a clear relation between the property and performance metric exists. Accurate performance models, however, will be a function of more complex groups of properties.

4.2 Temporal Density

The register area depends on both the number of registers, and the area of each. The latter is a function of the required bitwidth. This section concentrates on the description of the *temporal density* property measure for predicting the number of registers.

The temporal density measures the maximum expected number of variables that must be simultaneously kept alive. The first step involves estimating the variable lifetimes. The operator execution times determine the lifetimes of the variables, but are not known until scheduling is performed (at which point the left edge algorithm [Kur87] could be used to determine the number of registers). At this level, however, a heuristic measure is defined instead, which approximates the expected lifetime distributions of the variables in the flow graph. For these calculations, operations are assumed to have equal probability of executing in any of its valid time slots. The lifetime of each variable v is approximated from the expectation of the execution times of its production and consumption nodes. The expected start and end times are calculated using the following formulas:

$$\text{ExpectedStart}_v = 0.5 \times (\text{ASAP}_x + \text{ALAP}_x) + \text{Duration}_x , \quad (1)$$

$$\text{ExpectedEnd}_v = \max_{y_i \in y} (0.5 \times (\text{ASAP}_{y_i} + \text{ALAP}_{y_i}) + \text{Duration}_{y_i}) \quad (2)$$

where x is the producer node, and $y_i \in y$ are the consumer nodes.

A cyclic lifetime graph displaying the lifetimes of all variables in the graph is then constructed. The temporal density measure is defined to be the maximum number of variables that cross a vertical line situated at any time t.

An empirical study was conducted to detect the trends between actual numbers of registers, and the temporal density measure. The scatter plot of Figure 5 clearly indicates a strong correlation between temporal density and the number of registers

in the final implementation. Accurate performance models are likely to be a function of just the temporal density measure.

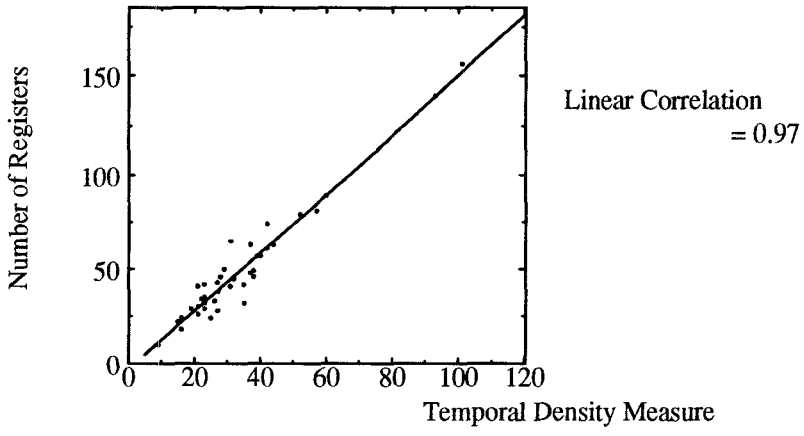


Figure 5: Number of Registers versus Temporal Density Measure for 40 Examples

4.3 Regularity

Often the largest component of semi-custom ASIC implementations is the interconnect area, which consists of not only wiring, but also the supporting muxes, buffers, and white space introduced. This area depends on the bitwidths of the datapaths, number of each interconnect element, and amenability of the design to compact routing. Several algorithmic properties are good predictors of the later 2 elements of interconnect area. These are regularity, which is related to the descriptive complexity of the graph, operator concurrency, and interconnect concurrency, which is similar to concurrency of execution unit accesses considered previously. Concurrency for interconnect can be derived by substituting transfer operations for execution units in the previous treatment of Section 4.1. This section will concentrate on the introduction and description of the *regularity* property measure.

The influence of regularity on physical layout characteristics has been observed by several high level synthesis groups [Not91, Rao92]. Regularity has only been treated qualitatively until now, and the exploration of its potential is in an early phase.We define regularity using the following simple formula:

$$\text{Regularity} = \frac{\text{Size}}{\text{Descriptive Complexity}} \tag{3}$$

The *size* component is the number of operations and data transfers executed in the computation. The *descriptive complexity* is a measure of the shortest description from which the graph can be reproduced. A regular graph has repeated patterns of nodes and interconnections, and can thus be concisely described. The inverse proportional relation is defined since lower complexity implies higher regularity. It is also necessary to consider how complexity scales with the size of the computation.

The proportional relation is established since complexity is a function of the size of the graph being described.

Motivated by the underlying ideas in descriptive complexity of strings (Kolmogorov complexity) [Li90], a measure of graph complexity has been developed. Given an encoding scheme (language by which to describe the graph and a measure of the length of a program), the complexity of a graph is defined as the length of the shortest program which can represent the graph.

Description of a graph involves describing the types and numbers of nodes, and the exact interconnections between them. A pseudo-descriptive language was developed for this purpose. The pseudo language has 2 primary instructions: instantiate and connect. The instantiate instruction takes 2 parameters - the operation to instantiate, and the number of instances of that operator that are desired. If the operator is not a primitive of the language (i.e. templates, loops, subgraphs), then the operator must be described using basic elements of the language.

For a given graph, there exist a number of programs that can describe it, some being more compact than others. Consider the graph of Figure 6. The Figures 6a - 6c show various descriptive versions of this graph, where the degree to which a subroutine is used is indicated on the corresponding graphs. In each of the flow graphs, a different number of patterns has been detected. A sample descriptive program is shown for the second. The length of a program in this pseudo-descriptive

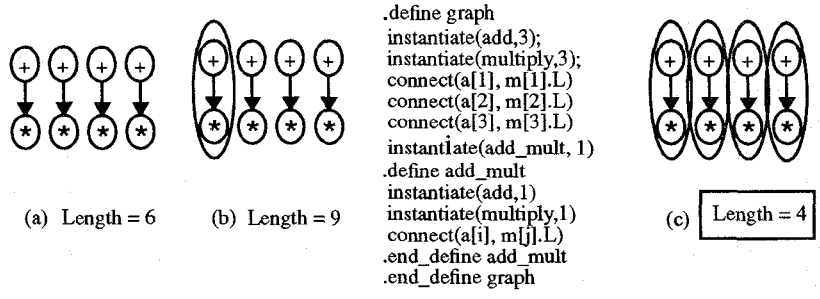
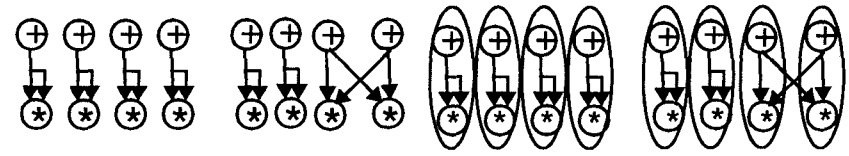


Figure 6: Example of Language and Program Length

language is defined as the number of instantiate and connect statements. The original cost of describing the graph if no patterns are detected is 6. The overhead associated with detecting only a single pattern increases the cost to 9. The lowest cost is 4 (Figure 6c), which is the complexity of the graph. The example of Figure 7 emphasizes the fact that the majority of complexity in graph descriptions is due to the description of the interconnect structure. Both graphs of Figure 7a and 7b have the same types and numbers of nodes. The flat description of either graph is 10. The interconnect structure of Figure 7a is much more regular, however, and thus allows the efficient description of the graph.

Template matching techniques are clearly the tool of choice for the detection of the common patterns that enable a concise description of a graph. The template covering involves both template generation and hierarchical template covering. The



(a) Length = 10 (b)Length = 10 (c) Length = 5 (d) Length = 11

Figure 7: Encoding Example: Majority of Complexity is in Interconnect Structure

objective function for the template selection is the encoding cost. The covering is done hierarchically to capture the regularity at all levels of granularity.

Several examples that illustrate the underlying influence of regularity are presented next. The intuition is that high regularity corresponds to simple description of the interconnections in the computation, which can translate into *fewer* types of interconnections. First consider the two IIR filters of Figure 8. Both can be imple-

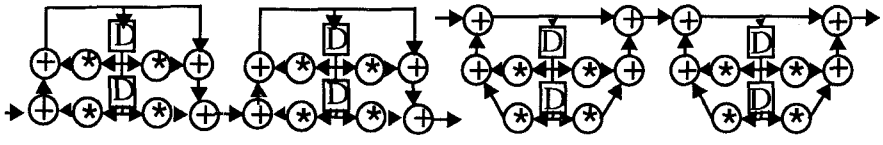


Figure 8: 4th order IIR structures: a) Regular version; b) Less Regular version

Structure	Regularity	Size	Number of Busses	Number of Muxes
Less regular version	2.45	49	8	4
Regular version	2.72	49	6	2

Table 1: Regularity for Interconnect Area - Two versions of a 4th order IIR filter

mented for a throughput of 10 control steps using 1 multiplier, 1 adder, and 1 transfer unit. Interconnect requirements are shown in Table 1. The more regular version indeed results in less busses and muxes. Consider next the well-known wave digital

Structure	Regularity	Size	Number of Busses	Number of Muxes	Number of Buffers
8th order Wave Digital	1.7	127	6	5	11
8th order Ladder	3.5	132	4	4	5

Table 2: Regularity for Interconnect Area - wave digital versus ladder structures

and ladder IIR filter structures. For filters of the same order, both have approximately the same size. The interconnect requirements are shown in Table 2. The more orderly network of communications that is present in the ladder structure

translates into a simpler overall interconnect structure, regardless of its slightly larger size.

5. CONCLUSIONS

This paper has presented a framework for system-level design guidance which is founded upon characterization of algorithms by their properties. The case study has illustrated relations that exist between ASIC area components and algorithmic properties. In this context, the concurrency, temporal density, and regularity properties were presented. The presented work forms a foundation for extension to other design metrics. For example, area prediction is in fact crucial for accurate power prediction. Most of our work to date has concentrated on custom-ASIC implementations. The methodology is the same however, in generalization to other implementation platforms: for a given metric on a particular platform, a few relevant properties are combined to construct a prediction as feedback to the user.

ACKNOWLEDGEMENTS

This project is sponsored by a fellowship from AT&T and the Office of Naval Research as well as ARPA grant J-FBI 93-153.

REFERENCES

[Amd64] G.Amdahl, G.Blaauw, F.Brooks, "Architecture of the IBM System/360," *IBM Journal of R&D*,Vol. 8,No. 2,pp. 87-101, 1964.

[Jam87] L. Jamieson, "Characterizing Parallel Algorithms," in *The Characteristics of parallel algorithms*, L. Jamieson, D. Gannon, R. Douglass (eds.), MIT Press, Cambridge, Mass., 1987.

[Knu71] D. Knuth, "An Empirical study of FORTRAN programs," *Software Practice & Experience*, Vol. 1, No. 2, pp. 105-133, 1971.

[Kun88] S.Y. Kung, *VLSI Array Processors*, Prentice Hall, Englewood Cliffs, NJ, 1988.

[Kur87] F. Kurdahi, A. Parker, "REAL: A Program for REgister ALlocation," *Proc. 24th DAC*, pp. 210-215, 1987.

[Li90] M. Li, P. Vitanyi, "Kolmogorov Complexity and its Applications," in *Handbook of Theoretical Computer Science*, Jan van Leeuwen (ed.), pp. 187-254, MIT Press, Cambridge, MA, 1990.

[Lid94] D. Lidsky, J. Rabaey, "Low Power Design of Memory Intensive Functions Case Study: Vector Quantization," in *VLSI Signal Processing, VII*, IEEE Press, New York, 1994.

[Not91] S. Note, et al., "Cathedral-III: Architecture-Driven High-Level Synthesis for High Throughput DSP Applications," *Proc. 28th DAC*, pp. 597-602, 1991.

[Pap93] E. Papaefstathiou, D. Kerbyson, G. Nudd, "A Layered Approach to the Characterization of Parallel Systems for Performance Prediction," *Performance Evaluation of Parallel Systems Workshop*, Coventry, UK, pp. 26-34, 1993.

[Pau89] P. Paulin, J. Knight, "Force-Directed Scheduling for the Behavioral Synthesis of ASIC's," *IEEE Trans. on CAD*, Vol. 8, No. 6, pp. 661-679, 1989.

[Pot93] M. Potkonjak, J. Rabaey, "Exploring the DSP Algorithm Design Space using HYPER," in *VLSI Signal Processing, VI*, L. Eggermont et al., (eds.), IEEE Press, New York, pp. 123-131, 1993.

[Rab91] J. Rabaey, C. Chu, P. Hoang, M. Potkonjak, "Fast Prototyping of Data Path Intensive Architectures," *IEEE Design & Test Magazine*, Vol. 8, No. 2, pp. 40-51, 1991.

[Rao92] D. Rao, F. Kurdahi, "Partitioning by Regularity Extraction," *Proc. 29th DAC*, pp. 235-238, 1992.

[Tho93] D. Thomas, J. Adams, H. Schmitt, "A Model and Methodology for Hardware-Software Codesign," *IEEE Design & Test of Computers*, Vol. 10, No. 3, pp. 6-15, 1993.