Aspects of system modelling in Hardware/Software partitioning

Knudsen, Peter Voigt; Madsen, Jan

Published in:
Proceedings. Seventh IEEE International Workshop on Rapid System Prototyping

Link to article, DOI:
10.1109/IWRSP.1996.506721

Publication date:
1996

Document Version
Publisher's PDF, also known as Version of record

Aspects of System Modelling in Hardware/Software Partitioning

Peter Voigt Knudsen and Jan Madsen
Department of Information Technology, Technical University of Denmark
pvk@it.dtu.dk, jan@it.dtu.dk

Abstract

This paper addresses fundamental aspects of system modelling and partitioning algorithms in the area of Hardware/Software Codesign. Three basic system models for partitioning are presented and the consequences of partitioning according to each of these are analyzed. The analysis shows the importance of making a clear distinction between the model used for partitioning and the model used for evaluation. It also illustrates the importance of having a realistic hardware model such that hardware sharing can be taken into account. Finally, the importance of integrating scheduling and allocation with partitioning is demonstrated.

1 Introduction

Hardware/software partitioning is often viewed as the synthesis of an architecture consisting of a single CPU and a single dedicated hardware component (full custom, FPGA, etc.) from an initial system specification, e.g., [1].

The aim of this paper is to emphasize the importance of clearly defining and reporting the partitioning model assumed by a partitioning algorithm and to demonstrate the kind of errors that may occur if the results produced by a partitioning algorithm which assumes a simplistic partitioning model are not evaluated in accordance with a realistic implementation oriented model. This will make it easier to evaluate particular partitioning approaches and to compare strong sides and weaknesses of different approaches.

Even though the single CPU, single ASIC architecture is a special and limited example of a distributed system, the architecture is relevant in many areas such as DSP design, construction of embedded systems, software execution acceleration and hardware emulation and prototyping [8]. Further, it is the most commonly used target architecture for automatic hardware/software partitioning approaches.

The hardware/software partitioning of a system specification onto a single CPU, single ASIC architecture has been investigated by a number of research groups [1, 2, 3, 5, 6, 9] which have employed widely different input languages (C, C^x, VHDL, etc.) and system models, and have had different optimization goals and constraints in mind. This makes it very difficult to compare the approaches. The different approaches will be described and discussed in section 3 and 4.

2 Modeling and Evaluation Aspects

In order to solve a problem by means of computer tools, it is necessary to build a model of the real world problem, i.e., to transform the problem from the physical domain into the model domain. In doing so, a number of details are disregarded as to keep the model simple, however, the main characteristics of the physical problem should still be present. When a suitable model has been found, the problem is solved within the model domain. Finally, the model domain result has to be realized in the physical domain. In figure 1 this is depicted for the partitioning problem.

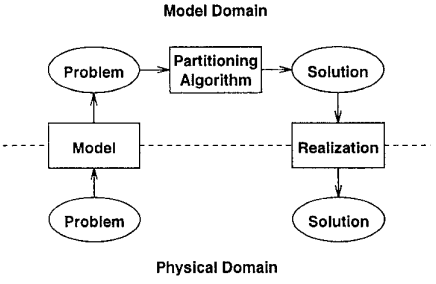

Figure 1: *Domain transformations.*

It is important to realize, that results which are optimal within the model domain may not be optimal when transformed to the physical domain. Clearly, it is important to model the real world as closely as possible and it is equally important to always carry out a series of tests in the real world domain when the performance and quality of an algorithm is evaluated. However, actually implementing the partitioned system in order to evaluate it is costly and time consuming, thus, it is important to be able to evaluate the different solutions in the model domain, before selecting the one to be implemented.

In general, the system model used by the partitioning algorithm may deviate from how the system is implemented in reality. In the following, the phrase *partitioning model* is used to denote the model used by the partitioning algorithm, and the phrase *realization model* is used to denote the model of the target architecture, i.e., the way in which a partition is assumed to be implemented.

Figure 2 shows two ways of evaluating a partitioning algorithm within the model domain:

0-8186-7603-5/96 $5.00 © 1996 IEEE

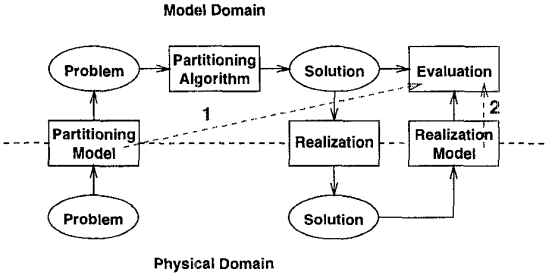

Figure 2: *Evaluating the partitioning result in the model domain; 1) Evaluating according to the partitioning model, 2) Evaluating according to the realization model.*

1. Evaluation according to the partitioning model.
2. Evaluation according to the realization model.

A reason for choosing a simple partitioning model is that some partitioning algorithms can then solve the constrained partitioning problem faster and/or more elegantly. But, as this paper will emphasize, the quality of the algorithm *must* be evaluated according to the realization model which is actually chosen for implementation in the physical domain.

3 Partitioning Models

In order to be able to partition a system, it has to be divided into fragments, in the following called Basic Scheduling Blocks or BSBs. The BSBs define the granularity of the partitioning, and are the chunks of code which have to be mapped to software or hardware.

In this section we present three of the most commonly used partitioning models. Note that many other and more advanced partitioning models than the ones presented here can be thought of (models including global communication, function calls, hardware and software processes executing in parallel, global optimizations, etc.). The purpose of listing the following partitioning models is, however, not to present a general classification scheme for partitioning approaches but rather to be able to reach some general conclusions which will be valid no matter which model a particular partitioning algorithm is based on.

The presented models are arranged by increasing complexity and all models are based on a system specification which is represented as a sequence of BSBs:

Definition 1 *A system specification,*

$$S = \langle B_1, B_2, \ldots, B_n \rangle$$

is an ordered list of n BSBs, where B_i denotes BSB number i and BSBs are ordered according to the execution sequence.

It is assumed that all function calls have been flattened prior to BSB extraction. Also, it is assumed that BSBs execute under mutual exclusion, i.e., hardware and software BSBs are not allowed to execute simultaneously.

Definition 2 *A BSB, B_i, is defined by the six-tuple,*

$$B_i = \langle a_{s,i}, t_{s,i}, a_{h,i}, t_{h,i}, r_i, w_i \rangle$$

where $a_{s,i}$ and $t_{s,i}$ are the area and execution time of B_i when placed in software, $a_{h,i}$ and $t_{h,i}$ are the area and execution time of B_i when placed in hardware, r_i and w_i are the read-set and write-set variables of B_i, respectively.

The software area $a_{s,i}$, which indicates the software object code size, is not considered in the following discussions (infinite software side capacity is assumed) and is only included for completeness. It is assumed that $a_{s,i}, t_{s,i}, a_{h,i}$ and $t_{h,i}$ are all *fixed* values calculated prior to partitioning, i.e., each BSB has a fixed hardware area $a_{h,i}$ and execution time $t_{h,i}$ independent of which other BSBs are implemented in hardware, and likewise for the software implementation. In section 6 these assumptions will be elaborated upon. We will use the notation $t(r_i)$ and $t(w_i)$ to denote the communication time needed to transfer the variables of the read-set and write-set to/from B_i respectively. Area contributions associated with communication primitives are ignored in this paper.

Finally, let a partition be defined as,

Definition 3 *A partitioning of a system specification, $S = \langle B_1, B_2, \ldots, B_n \rangle$ is a mapping $P : \langle B_1, B_2, \ldots, B_n \rangle \rightarrow \{SW, HW\}$.*

Model 1: Instantaneous Communication

The simplest possible model assumes instantaneous communication, i.e., $t(r_i) = t(w_i) = 0$. Figure 3 shows the initial all-software solution and a solution after partitioning.

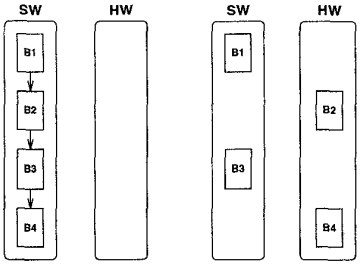

Figure 3: *Partitioning model with instantaneous communication.*

In this model each BSB, B_i, receives a constant (possibly negative) speedup[1], $s_i = t_{s,i} - t_{h,i}$, when implemented in hardware, independent of which other BSBs

[1]In this paper we will use the term "speedup" to denote the time saved by moving functionality to hardware.

are implemented in hardware. Hence, the total speedup obtained by the partition may be defined as

$$s = \sum_{B_i \in HW} s_i$$

Model 2: Simple Communication

A simple way of taking communication into account is depicted in figure 4a.

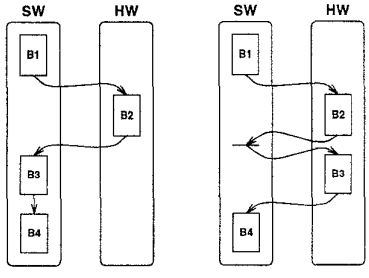

Figure 4: *Partitioning model with simple communication model.*

In this model, if a BSB is placed in hardware, its read-set variables are always transferred from software to hardware prior to execution of the BSB, and its write-set variables are always transferred back to software when it has finished its execution, regardless of how other BSBs are placed. This model allows us to associate a *fixed* hardware execution time, including communication time, with each BSB. Hence, the total speedup obtained by the partition may be defined as

$$s = \sum_{B_i \in HW} (t_{s,i} - (t(r_i) + t_{h,i} + t(w_i)))$$

where $t(r_i)$ and $t(w_i)$ are the communication times needed to transfer the variables of the read-set and the write-set to/from B_i.

Figure 4b illustrates the main problem of the simple communication model. It is evident that there is an unnecessary communication overhead associated with the communication from B_2 to B_3 which results in an implementation which is not optimal. However, in this simple model where we have to be able to associate a fixed hardware execution time with each BSB, that is inevitable.

Model 3: Adjacent Block Communication

Considering figure 4b it would be better if B_2 could send its write set variables directly to B_3 by storing them in local hardware memory. This is the main idea of model 3 and is depicted in figure 5a.

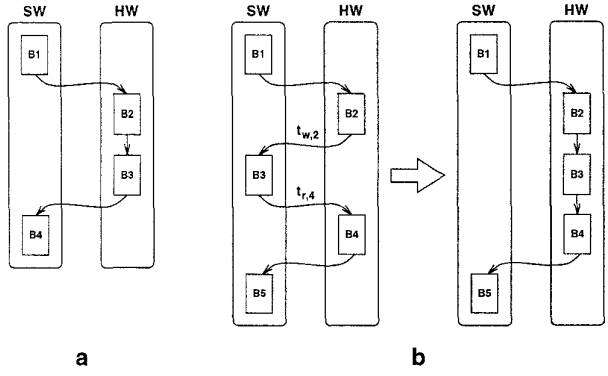

Figure 5: *Partitioning model with adjacent block communication.*

The adjacent block communication gives a more realistic model of the partitioned system. However, it is no longer possible to associate a fixed execution time with each BSB, as the execution time now depends on whether the previous and following blocks are placed in hardware. I.e., each time a BSB is moved to hardware, an analysis of the influence on the read/write sets has to be performed. This also makes the calculation of the speedup more difficult, as it now depends on the sequence of BSBs placed in hardware and the communication through their *effective* read- and write-sets.

Figure 5b illustrates how the model can lead to a better partitioning result than the previous models. B_3 is a BSB which receives no speedup when transferred to hardware. Within model 1 and 2 it will therefore be placed in software. But within model 3 placing it in hardware will result in a speedup as $t_{w,2}$ and $t_{r,4}$ disappear, so this may be done unless the area penalty is too large compared with the speedup.

4 Partitioning Approaches

From the preceding discussions we may define two constrained partitioning problems:

1. Optimize for speed with an area constraint.
2. Optimize for area with a speedup constraint (which equals the total software execution time minus an execution speed constraint).

Approaches Assuming Instantaneous Communication

The system model presented by Jantsch et al. [5] corresponds to partition model 1 as communication is ignored and BSBs cannot execute in parallel.

The presented results are model domain results based on a realization model equal to model 1, i.e., no attempts to evaluate according to a realistic realization model have been made. Thus, the reported results may be overly optimistic.

Approaches Assuming Adjacent Block Communication

The COSYMA system by Henkel, Ernst et al. [1, 2, 4] uses a simulated annealing algorithm which indirectly attempts to minimize hardware (as no explicit hardware area costs are calculated) given several local execution-time constraints. The algorithm accounts for adjacent block communication optimization, so the system model corresponds to partitioning model 3. The results which are reported are physical domain results measured on a system implementation which is also based on model 3.

In [7] we present a dynamic programming algorithm called PACE which takes adjacent block communication into account during partitioning. As for the COSYMA system this approach assumes that hardware and software BSBs cannot execute in parallel.

Other Approaches

Some approaches do not present their partitioning model together with the presentation of the algorithm and results. This makes it difficult to judge upon the quality of the algorithm. Vahid et al. [9] is an example.

Other approaches use a partitioning model which does not fit nicely into the simple models we have presented. For instance, Kalavade and Lee [6] and Gupta and De Micheli [3] both present a partitioning approach which uses a model that resembles partitioning model 3 but differs in that hardware and software BSBs may execute in parallel. [3] presents physical domain evaluations while [6] uses model domain evaluations.

5 Experiments with Different Partitioning and Evaluation Models

A series of experiments which demonstrate the effects of optimizing according to one model and evaluating according to another model have been carried out in order to be able to discuss and compare the partitioning approaches presented in the previous section. The sample application used for the experiments is a VHDL behavioral description consisting of 372 lines code. The VHDL program is translated into a control/dataflow graph (CDFG) and divided into a number of BSBs. The CDFG contains 1043 vertices, 1143 edges, 8 loops (nested up to three levels) and one if-then-else construct.

For the experiments, software execution-time estimates are based on a M68000 microprocessor, and hardware execution-time estimates are obtained from a global schedule of the CDFG onto an Actel ACT 3 FPGA.

The investigated partitioning problem is to optimize for speed with a hardware area constraint. A Knapsack algorithm is used to solve the partitioning problem for partitioning models 1 and 2. To solve the partitioning problem for partitioning model 3 the PACE [7] algorithm is used. For all tests, a fixed datapath allocation area of 1148 is assumed and the area of hardware BSBs is the estimated area of their hardware controller based on an Actel ACT 3 FPGA. Each logic/sequential module in the FPGA is assigned the area 1. Memory mapped I/O is assumed for hardware/software communication. Partitioning is performed for a sequence of available hardware areas ranging from 1000 to 2000 in steps of 20.

In the following figures, the names of the individual graphs have the form "Ax:Ey". This means that the graph shows the result of employing a partitioning algorithm which optimizes according to partitioning model x and evaluating the resulting partition according to realization model y (the models are numbered as in section 3). Only model domain evaluations have been carried out.

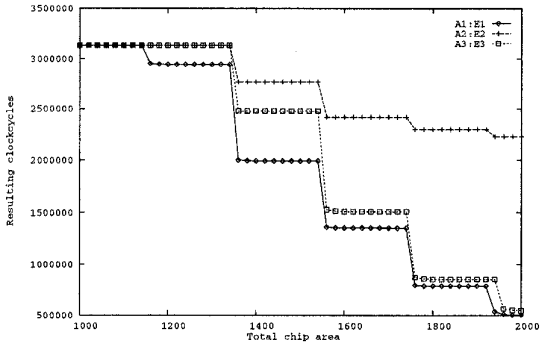

Figure 6: *Partitioning and evaluation according to the same model.*

Figure 6 illustrates the results of partitioning and evaluating according to the *same* model. As discussed in section 4 this is how most approaches are presented and related to each other.

For chip areas less that the allocated area for the datapath, no speedup is obtained as no control area is available. As soon as control area is available, the A1:E1 approach starts to move BSBs to hardware. For the approaches taking communication into account, moving BSBs to hardware is not beneficial until the total area reaches around 1370. It can be seen that as the chip area increases, more and more BSBs are moved to hardware, thus, for large chip areas there is less communication between hardware and software, hence the A3:E3 and A1:E1 approaches become comparable. Also, it is clear that the A2:E2 approach does not move as many BSBs to hardware as the two other approaches. This is mainly due to the fact that many of the BSBs have a communication overhead which is larger than the speedup they induce.

Just comparing the curves of figure 6 may lead to the conclusion that A1:E1 is the best approach. However, taking the partitions of figure 6 and just evaluating them according to the most realistic model (model 3) results in the curves shown in figure 7.

The first thing that is noted is, that even though the A1:E1 approach of figure 6 "thinks" it is achieving a large speedup for areas in the range 1200 to 1700,

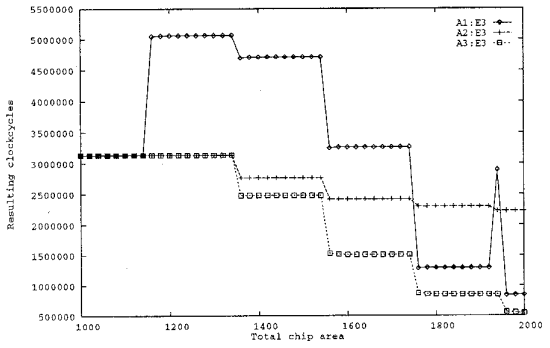

Figure 7: *Comparison of the results obtained by optimizing according to each of the three partitioning models and implementing according to the most realistic realization model 3 which optimizes adjacent block communication.*

it is actually producing worse than the all-software solution (the A1:E3 curve). Thus, using a partitioning approach which assumes instantaneous communication when evaluated, as in [5], the reported results may be wrong; in the experiments performed in our paper the approach was wrong by a factor of 2. It should also be noted that as the A1:E1 approach assumes instantaneous communication, it may, by pure chance, select BSBs which actually give a good partitioning, however, this will not be decidable.

6 Hardware Modeling and its Relation to Partitioning

In order to have realistic hardware area estimates it is important to also have a realistic hardware model. In the preceding discussion it was presumed that the hardware area of BSBs remained fixed independent of the partitioning. The hardware area of a BSB (B_i) may be viewed as containing two parts; a datapath area ($a_{dp,i}$) in which all computation (and storage) is performed, and a control unit area ($a_{cu,i}$) which controls the execution of the datapath, i.e., $a_{h,i} = a_{cu,i} + a_{dp,i}$.

There is a strong relationship between the amount (and type) of allocated hardware resources in the datapath ($a_{dp,i}$) and the hardware execution time, as more hardware resources allows for exploiting the inherent parallelism of the application. When an allocation has been selected, the execution time of a BSB is found by scheduling the operations in the BSB. The scheduling will also determine the area of the control unit ($a_{cu,i}$).

Now, a fixed hardware area ($a_{h,i}$) of a BSB (B_i) can be obtained in one of the following ways.

1. Hardware resources are allocated and scheduled separately for each BSB before partitioning. The total hardware area may be expressed as

$$a_h = \sum_{B_i \in HW} a_{cu,i} + \sum_{B_i \in HW} a_{dp,i}$$

In this way hardware sharing among BSBs is not considered.

2. Global allocation of hardware resources and individual scheduling of the BSBs is performed before partitioning. The fixed hardware area $a_{h,i}$ of the individual BSBs is then the area of their corresponding hardware controller plus the increase in multiplexer and interconnect area[2]. The available area for BSBs is equal to the total hardware area minus the area of the allocated hardware resources. In this case the total hardware area of a single BSB may be expressed as

$$a_h = a_{dp} + \sum_{B_i \in HW} a_{cu,i}$$

In the next section we will demonstrate the importance of being able to consider scheduling and allocation during partitioning. As the first approach is clearly inefficient, we will only consider the second method.

7 Experiments With Different Allocations

A series of experiments to demonstrate the effect of having different allocations (and thus different schedules) have been carried out.

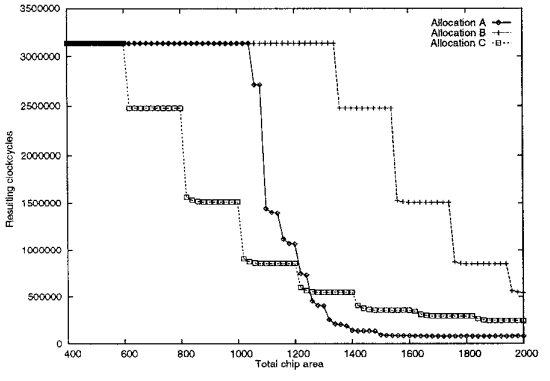

Figure 8: *Partitions with three different allocations.*

Figure 8 shows the results of partitioning the sample application of section 5 with the PACE algorithm using the three allocations A, B and C, listed in table 1.

Both partitioning and evaluation is done according to model 3, i.e., taking adjacent block communication into account, hence figure 8 shows A3:E3 curves.

[2]For the results presented in this paper, the multiplexer and interconnect areas are not considered.

Datapath allocations			
	Allocation		
Module	*A*	*B*	*C*
add-sub-comb	2	1	1
mul-comb	1	0	0
mul-ser	0	8	1
div-comb	1	0	0
div-ser	0	1	1
Area	760	1148	427

Table 1: *Modules and corresponding area for each of the three allocations.*

Hardware modules used for the allocation experiment			
Module	*Operations*	*Area*	*Clock-cycles*
add-sub-comb	add, subtract, less, equal	97	1
mul-comb	multiply	339	4
mul-ser	multiply	103	16
div-comb	divide	339	4
div-ser	divide	103	16

Table 2: *Area and execution-time estimates for hardware modules and operations.*

Table 2 summarizes the characteristics of the most important of the allocated hardware modules. All figures are estimates based on the Actel ACT 3 FPGA, and have been used to calculate the allocation areas shown in table 1.

As figure 8 shows, widely different results are obtained for given available areas, and a specific allocation which is optimal for all areas cannot be found.

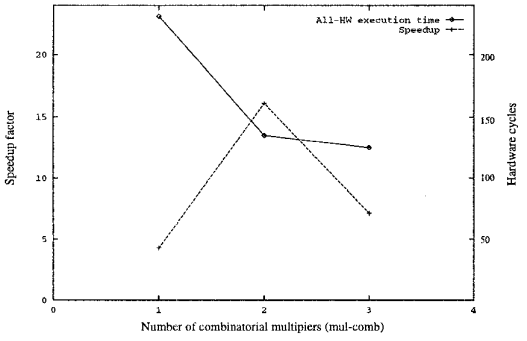

Figure 9: *Controller/datapath area trade-offs when partitioning under a hardware area constraint.*

Figure 9 illustrates this controller/datapath area trade-off when partitioning under hardware area constraints. The figure shows the all-hardware execution times for three different allocations, containing 1, 2 and 3 combinatorial multipliers respectively, and the corresponding obtained speedup. Going from 1 to 2 multipliers results in a significant speedup which would be expected as the all-hardware execution time is almost halved. However, the expected speedup when adding an extra multiplier (reducing the all-hardware execution time further) is *not* seen, instead the speedup is reduced to around the same level as for a single multiplier. The reason for this is of course that 3 multipliers allocate a large fraction of the available hardware area, leaving only a small area for the controller, and hence only a few BSBs can be moved to hardware.

8 Conclusion

Three basic partitioning models have been analyzed and discussed in relation to how they have been used by various research groups. We have shown the importance of making a clear distinction between partitioning models and evaluation models, i.e., the importance of making a realistic evaluation of results produced by algorithms operating on simple models. Thus, a clear distinction between partitioning model and realization model is a key issue for obtaining a basis for comparing different approaches.

Finally, we have shown the importance of integrating allocation and scheduling with partitioning and that this is particular important when partitioning under hardware area constraints, such as for FPGAs.

Acknowledgments

This work is supported by the Danish Technical Research Council under the "Codesign" program.

References

[1] R. Ernst, J. Henkel, and T. Benner. Hardware/software co-synthesis of microcontrollers. *Design and Test of Computers*, pages 64–75, December 1992.

[2] Rolf Ernst, Wei Ye, Thomas Benner, and Jörg Henkel. Fast timing analysis for hardware/software co-design. In *ICCD '93*, 1993.

[3] Rajesh K. Gupta and Giovanni De Micheli. System synthesis via hardware-software co-design. Technical Report CSL-TR-92-548, Computer Systems Laboratory, Stanford University, October 1992.

[4] D. Herrmann, J. Henkel, and R. Ernst. An approach to the adaptation of estimated cost parameters in the cosyma system. In *CODES '94*, 1994.

[5] Axel Jantsch, Peeter Ellervee, Johnny Öberg, Ahmed Hermani, and Hannu Tenhunen. Hardware/software partitioning and minimizing memory interface traffic. In *EURO-DAC '94*, 1994.

[6] Asawaree Kalavade and Edward A. Lee. A global criticality/local phase driven algorithm for the constrained hardware/software partitioning problem. In *Codes/CASHE'94*, pages 42–48, September 1994.

[7] Peter V. Knudsen and Jan Madsen. PACE: A dynamic programming algorithm for hardware/software partitioning. In *Codes/CASHE'96*, March 1996.

[8] Giovanni De Micheli. Computer-aided hardware-software codesign. *IEEE Micro*, 14(4):10–16, August 1994.

[9] Frank Vahid, Jie Gong, and Daniel D. Gajski. A binary-constraint search algorithm for minimizing hardware during hardware/software partitioning. In *EURO-DAC '94*, pages 214–219, 1994.