Hardware/Software Partitioning of VHDL System Specifications

Petru Eles[1,2], Zebo Peng[1], Krzysztof Kuchcinski[1], and Alexa Doboli[2]

[1] Dept. of Computer and Information Science
Linköping University
Sweden

[2] Computer Science and Engineering Department
Technical University of Timisoara
Romania

Abstract

This paper presents an approach for system level specification and hardware/software partitioning with VHDL. The implications of using VHDL as a specification language are discussed and a message passing mechanism is proposed for process interaction. We define the metric values for partitioning and develop a cost function that guides our heuristics towards performance optimization under hardware and software cost constraints. Experimental results are presented.

1. Introduction

In the last years new tools have been emerging which support the integrated design of both the hardware and the software component of a system. The input specification accepted by such a design tool describes the functionality of the application together with some design constraints and is typically given as a set of interacting processes.

Several approaches have been presented in the literature for the specification and partitioning of complex hardware/software systems. In [1, 8, 10] automatic partitioning is performed, while the approach presented in [12] is based on manual partitioning. Partitioning at a fine grain level is performed in [1, 8, 10]. In [1] and [13] partitioning is performed at a coarser granularity.

Iterative improvement algorithms are widely used for hardware/software partitioning. They are often based on simulated annealing [1, 8, 16], which is mainly due to the fact that simulated annealing algorithms can be quickly implemented and are widely applicable to different problems. However, a limitation of this method is the relatively long execution time and the large amount of experiments needed to tune the algorithm.

Several languages have been used for system modeling in hardware/software co-design environments. The approach described in [8] starts from an initial specification in C. Verilog is accepted as input language in [1] while the system described in [5] starts from VHDL specifications. Other systems are based on specifications captured in SpecCharts [18], ESTEREL [3], Statecharts [2], or SDL [12]. These choices are partially motivated by the particular application areas targeted by the different co-design systems, by the availability of specific software tools as part of the co-design environments and also by the experience and preferences of the respective research groups.

In our approach we start from a system specification in VHDL which has been extended for system-level design. Our goal is to synthesize a system which provides maximal performance (in terms of execution speed) using a given amount of hardware and software resources. The partitioning strategy is based on metric values derived from profiling (simulation), static analysis of the specification, and cost estimations and is performed automatically. We consider that minimization of communication cost between the software and the hardware partition and improvement of the overall parallelism are of outstanding importance during partitioning at system level.

We have implemented first a simulated annealing based algorithm for hardware/software partitioning. We then implemented our partitioning algorithm using the tabu search method. Based on extensive experiments we show that tabu search clearly outperforms simulated annealing.

The paper is divided into 6 sections. Section 2 introduces our hardware/software co-synthesis environment. Section 3 discusses features of VHDL as a system level specification language. In section 4 we are focusing on the successive steps of our partitioning strategy. The cost function and the evaluation of the partitioning algorithms is presented in section 5. Finally, section 6 presents our conclusions.

2. The co-synthesis environment

An overview of our hardware/software co-synthesis environment is depicted in Fig. 1. The input specification describes system functionality in terms of interacting processes modeled in VHDL which has been extended with a message passing mechanism for interprocess communication.

When the final partitioning is done, the hardware implementation is synthesized by the CAMAD high-level synthesis (HLS) system [17] while the software is generated

This work has been partially sponsored by the Swedish National Board for Industrial and Technical Development (NUTEK).

EURO-DAC '96 with EURO-VHDL '96
0-89791-848-7/96 $4.00

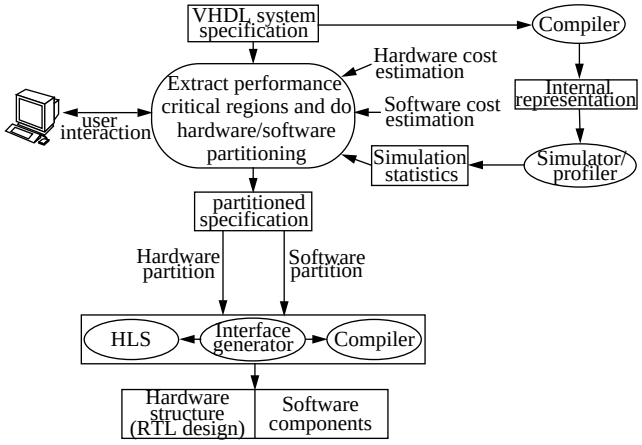

Fig. 1. Overview of the co-synthesis environment

by our VHDL to C compiler. We have made the following assumptions concerning the target architecture:

1. There is a single microprocessor executing the software part;
2. The microprocessor and the hardware coprocessor are working in parallel;
3. Reducing the amount of communication between the microprocessor and the hardware coprocessor improves the overall performance of the system.

This paper concentrates on the VHDL front-end of the co-synthesis environment, which performs extraction of performance critical regions and hardware/software partitioning.

3. VHDL for system-level specification

Complex systems are often specified as a set of interacting subsystems, each of which is described by a sequential process. VHDL [11] has several features that make it appropriate as a language for system level specification: data and control abstraction; structural hierarchy; concurrency, synchronization, and communication; timing specification; support for top-down design methodology; executable specifications (simulation); support for both hardware and software description.

Even though it is defined as a hardware description language, VHDL inherits features of Ada and thus includes constructs appropriate for the description of software. Benefiting from the above-listed features, the designer can specify complex systems in VHDL, without having to consider the possible implementation of components in hardware or software. It is also possible to execute this specification in order to test the resulted behavior and to collect some statistics that can be used to guide further design steps. There are also pragmatic arguments for using VHDL as a system-level specification language for co-synthesis. For example, there is a great number of commercially available VHDL simulation and hardware synthesis tools which can be integrated into new co-synthesis environments.

Accepting VHDL as the input language for a co-synthesis system requires that across all design steps toward the final synthesis result, semantic equivalence with the initial specification and its simulation behavior should be preserved. The main difficulty, in this context, concerns process interaction. According to the VHDL standard, synchronization and communication between processes is solved using signal assignment and wait statements, the semantics of which is defined in terms of the VHDL simulation cycle. Thus, synthesis of a VHDL design with process interaction specified at the signal level requires practically the implementation of the simulation cycle in order to achieve semantic correspondence between the specified model and the synthesized system [6]. However, such a low level synchronization and communication mechanism makes reasoning about processes and their interaction extremely complex. Therefore, both partitioning and synthesis become extremely difficult and inefficient if VHDL signal assignments and wait statements are directly used.

What we need is a high level process interaction mechanism that allows reasoning about processes and their interfaces during partitioning and synthesis, and at the same time can be efficiently implemented both in hardware and in software.

In [6] we present a model for system-level specification of interacting VHDL processes and describe the hardware synthesis strategy that we have developed for it. This model also fits to the framework of our hardware/software co-synthesis environment. According to the model, processes are the basic modules of the design, and they interact using a synchronous message passing mechanism with predefined *send*/*receive* commands. Communication channels are represented by VHDL signals. Assignment of a value to a signal is done by a *send* command. Processes that refer to a signal will wait until a value is assigned to it, by calling a *receive* command. Both *send* and *receive* have the syntax of ordinary procedure calls. Designs formulated according to this model are translated by a simple preprocessor into standard VHDL models for simulation purpose.

Processes communicating according to this mechanism are loosely coupled and can be implemented without enforcing the strong synchronization implied by the VHDL simulation cycle [6]. At the same time, communication interfaces between processes can be easily established or modified dynamically during partitioning, when new processes are created or functionality is moved from one process to another.

In the context of our co-synthesis environment, a VHDL description corresponding to this system-level specification model can be simulated, partitioned, and synthesized into hardware and software components preserving semantic correspondence with the initial VHDL specification.

4. Refinement of the VHDL specification

Partitioning starts from an initial system specification described as a set of VHDL processes interacting through communication channels. This specification is further

refined by decomposing it into processes of smaller granularity which, finally, have to be partitioned for hardware and software implementation respectively. The main goal of partitioning is to maximize performance in terms of execution speed. In order to achieve this we try to distribute functionality between the software and the hardware partitions taking into account communication cost and overall parallelism of the synthesized system. Thus, the following three objectives are considered:

1. To identify *basic regions* (processes, subprograms, loops, and blocks of statements) responsible for most of the execution time, in order to be assigned to hardware;
2. To minimize communication between partitions;
3. To increase parallelism within the resulted system at the following three levels:
   - internal parallelism of each process assigned to hardware;
   - parallelism between processes assigned to hardware;
   - parallelism between the hardware coprocessor and the microprocessor.

The partitioning algorithm takes into account simulation statistics, information from static analysis of the source specification, and cost estimations. Statistics data are collected from simulation of an internal representation generated by our VHDL compiler, loading the system with sets of typical input stimuli. Two types of statistics are used for partitioning:

1. *Computation load* (*CL*) of a basic region is a quantitative measure of the total computation executed by that region, considering all its activations during the simulation process. It is expressed as the total number of operations (at the level of internal representation) executed inside that region, where each operation is weighted with a coefficient depending on its relative complexity: $CL_i = \sum_{op_j \in BR_i} N\_act_j \times \varphi_{op_j}$; where $N\text{-}act_j$ is the number of activations of operation op_j belonging to the basic region BR_i and φ_{op_j} is the weight associated to that operation. The *relative computation load* (*RCL*) of a block of statements, loop, or a subprogram is the computation load of the respective basic region divided by the computation load of the process the region belongs to. The relative computation load of a process is the computation load of that process divided by the total computation load of the system.
2. *Communication intensity* (*CI*) on a channel connecting two processes is expressed as the total number of send operations executed on the corresponding signal.

Hardware/software partitioning is performed in four steps:

1. *Extraction of basic regions*;
2. *Process graph generation*;
3. *Process graph partitioning;*
4. *Process merging*: During the first step one or several *child* processes are possibly extracted from a *parent* process. If, as result of step 3, some of the child processes are assigned to the same partition with their parent, they are, optionally, merged back together.

4. 1. Extraction of basic regions

During the first partitioning step VHDL processes are examined individually to identify regions that are responsible for most of the execution time spent inside a process. Candidate regions are typically loops and subprograms, but can also be blocks of statements with a high CL. The search for candidate regions is performed bottom-up, starting from the inner blocks, loops, and the subprograms that are not containing other basic regions. When a region has been identified for extraction, a new process is built. It has the functionality of the original block, loop, or subprogram while communication channels are established to the *parent* process. These channels will be built using the message passing mechanism described in section 3. It is essential that synchronization using this process interaction mechanism affects not all processes (this is the case of process interaction based on signals) but only the respective parent and child process involved in the specific communication. Due to this strictly local effect of the transformations performed for interface generation, the original semantics of the VHDL specification is preserved.

In the following example we illustrate the generation of processes embedding a loop and a subprogram that were identified for extraction inside the two VHDL processes given below:

```
P1: process                          P2: process
  . . .                                . . .
  LOOP_1: while X<K loop               procedure P(A: in INTEGER;
    . . .                                        B: out INTEGER) is
    X := ... C + K ... ;               begin
    . . .                                . . .
  end loop LOOP_1;                       B := ... A ... ;
  . . .                                  . . .
end process P1;                        end P;
                                     begin
                                       . . .
                                       P(7, Z);
                                       . . .
                                     end process P2;
```

Given that the RCLs of the loop and the subprogram in the above processes are greater than a threshold, two new processes, P1_LOOP_1 and P2_PROC_P, are generated to execute the loop LOOP_1 and procedure P respectively. Communication channels to and from the new processes are established according to the data dependence relationship. In the above example, signals S_P1_C, S_P1_K, S_P1_X_TO, and S_P1_X_FROM are introduced for communication between P1 and P1_LOOP_1, and signals S_P2_A, and S_P2_B for communication between P2 and P2_PROC_P.

As stated, one of the objectives considered during partitioning is to increase parallelism within the resulted system. At process generation we follow this goal by introducing additional parallelism, as far as data dependence allows, between parent and child process. This is achieved by moving statements of the parent process into the sequence between the send and the receive commands used for communication with the child. The new VHDL code, after process extraction, is as follows:

```
signal S_P1_C,S_P1_K,S_P1_X_TO, S_P1_X_FROM, S_P2_A,
       S_P2_B: INTEGER;
P1: process
   . . .
   send(S_P1_C,C,S_P1_X_TO,
        X,S_P1_K,K);
   . . .  -- additional parallelism
   receive(S_P1_X_FROM);
   X := S_P1_X_FROM;
   . . .
end process P1;
P1_LOOP_1: process
   variable X : INTEGER;
begin
   receive(S_P1_C,S_P1_X_TO,
           S_P1_K);
   X := S_P1_X_TO;
   LOOP_1: while X < S_P1_K loop
      . . .
      X := ... S_P1_C + S_P1_K ... ;
      . . .
   end loop LOOP_1;
   send(S_P1_X_FROM,X);
end process P1_LOOP_1;
```

```
P2: process
   . . .
   send(S_P2_A,7);
   . . .-- additional parallelism
   receive(S_P2_B);
   Z := S_P2_B;
   . . .
end process P2;
P2_PROC_P: process
   variable B: INTEGER;
begin
   receive(S_P2_A);
   . . .
   B := ... S_P2_A ... ;
   . . .
   send(S_P2_B,B);
end process P2_PROC_P;
```

The final decision if processes generated during this step are kept as separate modules or will eventually be merged back, depends on the two subsequent partitioning steps. An important criterion for this decision will be the intensity of communication between parent and child processes.

4. 2. Process graph generation

The input to the second partitioning step is a set of interacting VHDL processes. The data structure on which hardware/software partitioning is performed is the *process graph*. Each node in this graph corresponds to a process and an edge connects two nodes if and only if there exists at least one direct communication channel between the corresponding processes.

The graph partitioning algorithm takes into account weights associated to each node and edge. Node weights reflect the degree of suitability for hardware implementation of the corresponding process. Edge weights measure communication and mutual synchronization between processes. The weights capture simulation statistics and information extracted from static analysis of the system specification or of the internal representation resulted after its compilation. The following data extracted from static analysis are captured:

Nr_op_i: total number of operations in process i;

$Nr\_kind\_op_i$: number of different operations in process i;

L_path_i: length of the critical path (in terms of data dependency) through process i.

The weight assigned to process node i, has two components. The first one, $W1_i^N$, is equal to the CL of the respective process. The second one is calculated by the following formula:

$$W2_i^N = M^{CL} \times K_i^{CL} + M^U \times K_i^U + M^P \times K_i^P - M^{SO} \times K_i^{SO} ;$$

where:

K_i^{CL} is equal to the RCL of process i, and thus is a measure of the computation load;

$K_i^U = \frac{Nr\_op_i}{Nr\_kind\_op_i}$; K_i^U is a measure of the uniformity of operations in process i;

$K_i^P = \frac{Nr\_op_i}{L\_path_i}$; K_i^P is a measure of the potential parallelism inside process i;

$K_i^{SO} = \frac{\sum_{op_i \in SP_i} w_{op_j}}{Nr\_op_i}$; K_i^{SO} captures the suitability of operations of process i for software implementation. SP_i is the set of such operations (floating point computation, file access, pointer operations, recursive subprogram call, etc.) in process i and w_{op_j} is a weight associated to operation op_j, measuring the degree to which the operation has to be implemented in software.

The relation between the above-named coefficients K^{CL}, K^U, K^P, K^{SO} is regulated by four different weight-multipliers: M^{CL}, M^U, M^P, and M^{SO}, controlled by the designer.

Both components of the weight assigned to an edge connecting nodes i and j depend on the amount of communication between processes i and j. The first one is a measure of the total data quantity transferred between the two processes. The second one does not consider the number of bits transferred but only the degree of synchronization between the processes, expressed in the total number of mutual interactions they are involved in:

$$W1_{ij}^E = \sum_{c_k \in Ch_{ij}} wd_{c_k} \times CI_{c_k} ; \quad W2_{ij}^E = \sum_{c_k \in Ch_{ij}} CI_{c_k} ;$$

where Ch_{ij} is the set of signals (channels) used for communication between processes i and j; wd_{c_k} is the width of signal c_k in bits; CI_{c_k} is the communication intensity on signal c_k.

5. Process graph partitioning

After generation of the process graph hardware/software partitioning can be performed as a graph partitioning task. The partitioning information is captured as weights associated to the nodes and edges. These weights have to be combined into a cost function which guides the partitioning algorithm towards the desired objective.

5. 1. The cost function

The partitioning heuristics are guided by the following cost function which is to be minimized:

$$C(Hw,Sw) = Q1 \times \sum_{(ij) \in cut} W1_{ij}^E + Q2 \times \frac{\sum_{(i) \in Hw} \frac{\sum_{\exists (ij)} W2_{ij}^E}{W1_i^N}}{N_H} - Q3 \times \left(\frac{\sum_{i \in Hw} W2_i^N}{N_H} - \frac{\sum_{i \in Sw} W2_i^N}{N_S} \right) ;$$

where *Hw* and *Sw* are sets representing the hardware and the software partition respectively; N_H and N_S are the cardinality of the two sets*; cut* is the set of edges connecting the two partitions; *(ij)* is the edge connecting nodes *i* and *j*; *(i)* represents node *i*;

The partitioning objectives stated at the beginning of section 4 are captured by the three terms of the cost function:

- The *first term* captures the amount of communication between hardware and software partition. Decreasing this component reduces communication cost and also improves parallelism between processes in the hardware partition and those implemented in software.

- The *second term* stimulates placement into hardware of processes which have a reduced amount of interaction with the rest of the system relative to their computation load and, thus, are active most of the time. This strategy improves parallelism between processes inside the hardware partition where physical resources are allocated for real parallel execution. For a given process *i*, $\left(\sum_{\exists(ij)} W2_{ij}^{E}\right) / W1_{i}^{N}$ is the total amount of interaction the process is involved in, relative to its computation load. The whole term represents the average of this value over the nodes in the hardware partition.

- The *third term* in the cost function pushes processes with a high node weight into the hardware partition and those with a low node weight into the software one, by increasing the difference between the average weight of nodes in the two partitions. This is a basic objective of partitioning as it places time critical regions into hardware.

The criteria combined in the cost function are not orthogonal, and sometimes compete with each other. This competition between partitioning objectives is controlled by the designer through the cost multipliers *Q1, Q2*, and *Q3* which regulate the relative influence of the different metrics.

Minimization of the cost function has to be performed in the context of certain constraints. Thus, our heuristics have to produce a partitioning with a minimum for *C(Hw, Sw)* so that the total hardware and software cost is within some user specified limits:

$$\sum_{(i)\in Hw} H\_cost_i \le Max^H \quad ; \quad \sum_{(i)\in Sw} S\_cost_i \le Max^S \quad .$$

Cost estimation has to be performed before graph partitioning. In the current implementation of our environment, the CAMAD high level synthesis system [17] produces hardware cost estimations in terms of design area. Software cost, in terms of memory size, is estimated for each process through compilation by our VHDL to C compiler.

5. 2. Experimental results

As a final step of the hardware/software partitioning process the weighted graph is to be partitioned into two subgraphs so that design constraints are satisfied and the cost function is minimal.

Hardware/software partitioning, formulated as a graph partitioning problem, is NP complete. In order to efficiently explore the solution space, heuristics have to be developed which hopefully converge towards an optimal or near-optimal solution. We have implemented two such algorithms, one based on simulated annealing (SA) [14] and the other on tabu search (TS) [9]. The partitioning algorithms and the experimental strategy used for their evaluation are presented in [7]. We evaluated the partitioning algorithms based on extensive experimentation using 32 geometric and random graphs with number of nodes between 20 and 400. Here are the most important conclusions drawn from these experiments:

1. Near-optimal results can be produced by both algorithms.
2. SA is based on a random exploration of the neighborhood while TS is completely deterministic. The deterministic nature of TS makes experimental tuning of the algorithm and setting of the parameters less laborious than for SA. At the same time adaptation of the SA strategy for a particular problem is relatively easy and can be performed without a deep study of domain specific aspects. Development of a TS algorithm is more complex and has to consider particular aspects of the given problem.
3. Performances obtained with TS are excellent and definitely superior in comparison to those given by SA (on average more than 20 times faster), as shown in Fig. 2. The times represented in the figure are the average CPU times needed for partitioning for all graphs of the given dimension using the SA and TS based algorithm respectively[1]. This conclusion is very important especially in the context that no TS based hardware/ software partitioning approach has yet been reported, while SA continues to be one of the most popular approaches for automatic partitioning.

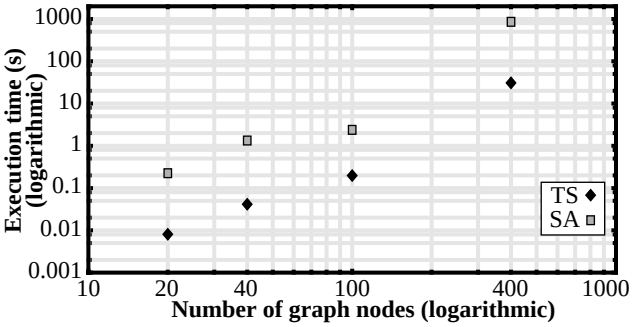

Fig. 2. Partitioning times with SA and TS

In order to validate our system level partitioning approach we performed two further experiments on real-life models: the *Ethernet network coprocessor* and the *OAM block of an ATM switch*. Both models were specified in

[1] The partitioning results were the same, both for SA and TS. For the 20 nodes graphs they were proven to be optimal by comparing with results obtained from exhaustive search. For the other graphs the good quality of the results was verified by comparing with results obtained from several very long and expansive runs with both TS and SA [7].

TABLE 1: Partitioning of the VHDL models

model	nr. of processes		part. with SA	part. with TS	t_{TS}/t_{SA}
	model	after extr.	t_{SA} (sec)	t_{TS} (sec)	
Eth. cop.	10	20	0.08	0.006	0.075
OAM bl.	19	27	0.10	0.007	0.07

VHDL. After simulation, basic regions were extracted and the annotated process graph has been generated. Partitioning was performed using both the SA and the TS based algorithm, with the cost function presented in section 5.1 and a constraint on the hardware cost representing 30% of the cost of a pure hardware implementation.

The *Ethernet network coprocessor* is given in [15] as an example for system specification in SpecCharts and has been used, in a HardwareC version, in [10]. We have rewritten it in VHDL, as a model consisting of 10 cooperating processes (730 lines of code). After the first partitioning step we got a VHDL specification consisting of 20 processes. The final partitioning produced a hardware partition with 14 processes and a software partition with 6 processes. The most time critical part of those processes that are handling transmission and reception of data on the ethernet line as well as processes which are strongly connected to them have been assigned to hardware and the rest belong to the software partition.

Our second example implements the *operation and maintenance (OAM) functions corresponding to the F4 level of the ATM protocol layer* [4]. We specified functionality as a VHDL model consisting of 19 interacting processes (1321 lines of code). The model resulted after extraction of basic regions has 27 processes. The resulted process graph has been partitioned into 14 processes assigned to hardware and 13 to software. Processes performing the filtering of input cells and those handling user cells (which constitute the overwhelming majority of received cells) were assigned to hardware. Processes handling exclusively OAM cells (which are arriving at a very low rate) were assigned to software.

In Table 1 we show the partitioning times using SA and TS for both examples which confirm the conclusions drawn from experiments with geometric and random graphs.

6. Conclusions

We have presented an approach to system-level specification and partitioning in hardware/software co-design. We have shown in this paper that such a specification can be captured by the VHDL language. While arguing the advantages of using VHDL, some limitation of VHDL as a hardware/software co-specification language has also been identified and a solution to resolve the problem is proposed. Our solution is based on the concept of reduced synchronization between VHDL processes, which is used to relax the strict synchronization imposed by the simulation-based semantics of VHDL.

We presented a technique to perform partitioning of the input VHDL specification, which supports the use of simulation results as well as static analysis of the input specification to guarantee the partitioning quality.

We formulated hardware/software partitioning as a graph partitioning problem and solved it by implementing two iterative improvement heuristics based on SA and TS respectively. The algorithms have been evaluated based on extensive experiments and the conclusions show that, while both can produce high quality solutions, TS is definitely superior in terms of partitioning time.

References

[1] J.K. Adams, D.E. Thomas, *Multiple-Process Behavioral Synthesis for Mixed Hardware-Software Systems*, Proc. Int. Symp. on Syst. Synth., September '95, IEEE CS Press, 10-15.

[2] K. Buchenrieder, C. Weith, *A Prototyping Environment for Control-Oriented HW/SW Systems Using State-Charts, Activity-Charts and FPGA's*, Proc. of EURO-DAC/VHDL'94, IEEE CS Press, 1994, 60-65.

[3] M. Chiodo, D. Engels, P. Guisto, H. Hsieh, A. Jurecska, L. Lavagno, K. Suzuki, A. Sangiovanni-Vincentelli, *A Case Study in Computer-Aided Co-design of Embedded Systems*, Design Automation for Embedded Systems, 1, 1996, 51-67..

[4] M. De Prycker, *Asynchronous Transfer Mode: Solution for Broadband ISDN*, Ellis Horwood, New York, 1993.

[5] W. Ecker, *Using VHDL for HW/SW Co-Specification*, Proc. of the EURO-DACVHDL'93, IEEE CS Press, 500-505.

[6] P. Eles, K. Kuchcinski, Z. Peng, M. Minea, *Synthesis of VHDL Concurrent Processes*, Proc. of EURO-DAC/VHDL'94, IEEE CS Press, 1994, 540-545.

[7] P. Eles, Z. Peng, K. Kuchcinski, A. Doboli, *Performance Guided System Level Hardware/Software Partitioning with Iterative Improvement Heuristics*, Res. Rep. LiTH-IDA-R-95-26, Dep. of Comp. and Inf. Science, Linköping University, 1995.

[8] R. Ernst, J. Henkel, T. Benner, *Hardware-Software Co-Synthesis for Microcontrollers*, IEEE Design & Test of Computers, September 1993, 64-75.

[9] Glover, E. Taillard, D. de Werra, *A User's Guide to Tabu Search*, Annals of Operations Research, vol. 41, 1993, 3-28.

[10] R.K. Gupta, G. De Micheli, *Hardware-Software Cosynthesis for Digital Systems*, IEEE Design & Test of Computers, September 1993, 29-41.

[11] *IEEE Standard VHDL Language Reference Manual*, IEEE Std 1076-1993, 1994.

[12] T. Ben Ismail, A.A. Jerraya, *Synthesis Steps and Design Models for Codesign*, Computer, February 1995, 44-52.

[13] A. Kalavade, E.A. Lee, *A Global Criticality/Local Phase Driven Algorithm for the Constrained Hardware/Software Partitioning Problem*, Proc. of Third International Workshop on Hardware/Software Codesign, IEEE CS Press, 1994, 42-48.

[14] S. Kirkpatrick, C.D. Gelatt, M.P. Vecchi, *Optimization by simulated annealing*, Science, vol. 220, no. 4598, 1983, 671-680.

[15] S. Narayan, F. Vahid, D.D. Gajski, *Modeling with SpecCharts*, Technical Report #90-20, Dept. of Inf. and Comp. Science, Univ. of California, Irvine, 1990/1992.

[16] Z. Peng, K. Kuchcinski, *An Algorithm for Partitioning of Application Specific Systems*, Proc. EDAC'93, IEEE CS Press, 1993, 316-321.

[17] Z. Peng, K. Kuchcinski, *Automated Transformation of Algorithms into Register-Transfer Level Implementation*, IEEE Trans. on Comp.-Aided Des. of Integr. Circ. and Syst., V13, no. 2, February 1994, 150-166.

[18] F. Vahid, S. Narayan, D. Gajski, *SpecCharts: A VHDL Front-End for Embedded Systems*, IEEE Trans. on Comp.-Aided Des. of Integr. Circ. and Syst., V14, no. 6, June 1995, 150-166.