A hardware/software partition methodology targeted to an FPGA/CPLD architecture

António J. Esteves and Alberto J. Proença
Department of Informatics, University of Minho
Braga, Portugal

Abstract

A two-step hardware/software partition methodology was developed. It departs from an initial partition solution based on the cluster growth algorithm and iteratively leads the designer to an improved solution, using the tabu search algorithm. A PCI-based reconfigurable architecture, EDgAR-2, was also developed, with an hybrid approach using both data path oriented devices (FPGAs) and control oriented ones (CPLDs). Two basic criteria were followed to evaluate the partition methodology in the design of embedded systems, targeting such hybrid reconfigurable architecture: the quality of the generated partition solutions and the accuracy of the estimates. Two data flow dominated case studies were selected: the cryptography algorithm DES and an image convolution with Sobel filter. The obtained results show that accurate estimates lead to high quality partition solutions.

Keywords: Partition methodology, PSM metamodel, tabu search, metrics estimation, evaluation

1 Introduction

This communication describes an automatic partition methodology oriented to develop data flow dominated, medium complexity and real time embedded systems, where a computer and an EDgAR-2 platform [1] form the reconfigurable architecture to be used in systems implementation. The partition task, implemented in the *parTiTool* tool, is part of a development methodology that covers all phases of systems development.

Partitioning is an NP-complete optimization problem that assigns system objects to the target architecture components and defines its startup time, to achieve the designer goals, quantified by a cost function. To reach this goal, the set of objects on the system description must be divided into a series of disjunct subsets that will be assigned to the different components of the target architecture. In the present work, the objects represent program-states or variables from the system PSM model. The present approach performs a functional, inter-component and automatic partition.

When compared to the methodology followed by the MOOSE approach [2], the proposed methodology has some advantages: (i) the state transition diagrams (STD) are replaced by PSM[1] models [3], which allow adequate handling of the system objects concurrency, (ii) implementations follow an iterative approach, replacing the traditional cascade design flow and (iii) the partition is automatically performed, without requiring additional expertise from a codesign professional.

To evaluate this partition methodology, *parTiTool* capabilities were compared to other approaches, following the structure introduced in [4]. Here, two sets of features are grouped for comparison purposes: the modelling support and the implementation support. The first identifies 3 axis: the application domain, the type of validation and the modelling style. Figure 1 shows where *parTiTool* fits in the graph and how it relates to other approaches. The proposed approach is part of a development methodology that uses heterogeneous modelling. It can be applied to data and control systems, but it is oriented to data flow dominated systems. In the present stage of the evolution, it does not allow co-verification.

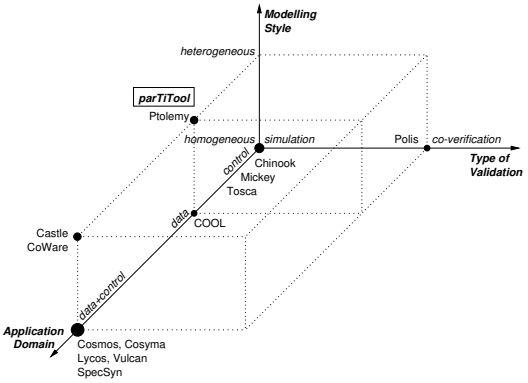

Figure 1: Categorization of approaches by modelling support.

To compare the support available to implement the systems with multiple components, figure 2 also uses 3 axis: the support to synthesize the interface between components, the supported target architecture and the automation degree of the partition process. A reasonable number of approaches (Chinook [5], Cosmos [6], CoWare [7] or Polis [8]) does not execute partition automatically. None of the approaches completely supports the automatic partition and the synthesis of

[1] **P**rogram-**S**tate **M**achine.

interfaces. In the proposed approach the partition process is automatic and the information required to synthesize the interface between components can be extracted from the detailed model used to estimate communication metrics.

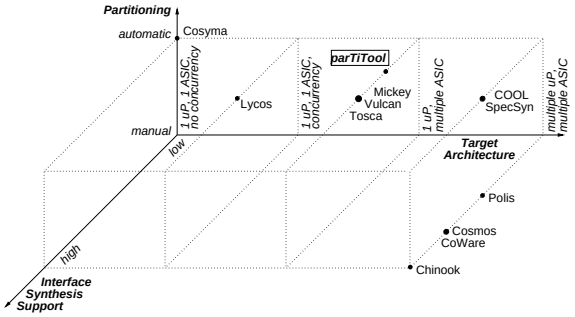

Figure 2: Categorization of approaches by implementation support.

The communication is organized in 6 sections. Section 2 summarizes the partition methodology under evaluation, describing the modelling that is relevant to the partition process, the creation and improvement of partition solutions with cluster growth and tabu search algorithms and the estimation of the metrics required by the evaluation functions. Section 3 introduces the target architecture. Section 4 presents two case studies used to evaluate the methodology and the results obtained with this evaluation are summarized in section 5. Finally, section 6 points out some conclusion remarks and directions for future work.

2 Partition methodology

The main modules of the partition methodology [9] [10] are identified in figure 3 and presented next: the partition algorithms, the evaluation functions and the metrics estimators.

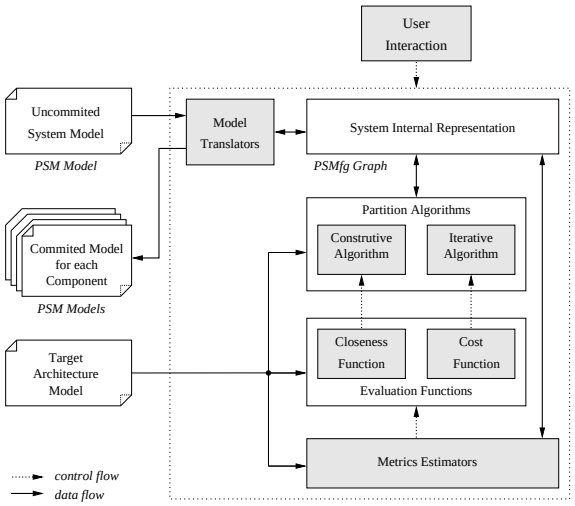

Figure 3: Structure of the partition methodology.

2.1 System modelling

In related approaches, the uncommitted systems are commonly modelled with meta-models such as CDFG, DFG, FSM, Petri net, CSP, an extended version of a previous meta-model, or a combination of these meta-models [11, 12, 13, 14, 2]. The PSM meta-model, chosen to describe the systems at the partition process interface, combines an HLL/HDL meta-model with HCFSM[2] [3]. A PSM model is described by a hierarchical set of program-states, where a program-state represents a computation unit, that at a given time can be active or inactive. A PSM model may include composite or leaf program-states. A composite program-state is defined by a set of concurrent or sequential program-substates, and a leaf program-state is defined by a block of code on the given programming language. The VHDL language was selected to describe the variables and the leaf program-states.

To describe systems during the partition process it was developed a CFG type meta-model: the PSM flow graph, or simply PSMfg. A PSMfg model is an acyclic, directed and polar graph. The PSMfg meta-model represents the semantic of PSM and all the information needed by the partition process, such as the metrics estimates and the assignment of objects to partitions. To control the granularity of the objects handled during the partition process, a PSMfg graph is able to represent the program-states structure. The computational support needed on PSMfg graphs edition and partition comes from the LEDA[3] library [15].

2.2 System partitioning

In the present work, partitioning is a two-step process: (i) create an initial partition solution with the cluster growth (CG) constructive algorithm and (ii) successively improve it with an iterative partition algorithm, such as tabu search (TS) or simulated annealing (SA).

At each iteration of the constructive process with the CG algorithm, the assignment of objects to partitions is guided by a function that measures the closeness among the object to assign and the previously assigned objects. On the definition of closeness it is considered the communication intensity between variables and program-states, the program-states computation time and the area occupied by variables and program-states in hardware. The used closeness function adapts itself to the type of object (variable or program-state) and partition (software or hardware).

The TS method, that successively improves the solution created by the CG algorithm, can be seen as an extension of the local search strategies, where a new solution is found on the present solution neighbourhood, applying a well defined set of rules [16] [17]. To reduce the set of the alternatives present on the current solution neighbourhood that will be evaluated, it is implemented a list with candidate solutions. The

[2] **H**ierarchical **C**oncurrent **F**inite **S**tate **M**achine.
[3] **L**ibrary of **E**fficient **D**ata types and **A**lgorithms.

search is further restricted by a set of forbidden (tabu) solutions.

When compared to the algorithm discussed in [16], the implemented tabu search algorithm [9] [10] only searches a partial neighbourhood of the present solution, has more evolution strategies to apply when there are no eligible solutions with quality and applies a more efficient improvement when no single move can further improve the cost of the present solution. The subset of moves that define the partial neighbourhood is made up the best move for each object of the system description. The implemented neighbourhood has a simple structure since a complex structure would greatly increase the computation time.

Given that the application of all tabu classifications can be very restrictive to the search, a subset of classifications was chosen that decreases the size of the neighbourhood to be searched and it does not place excessive restrictions to the search: (i) to classify as being tabu the move of an object from the source partition to the target partition, (ii) to classify as being tabu all the moves with the participation of that object and (iii) to classify as being tabu the inverse (move) of the move that originated the present solution. The implemented TS method includes two types of aspiration criterium: (i) it is applied an aspiration criterium by objective when the first alternative selects a move with quality that is classified as being tabu and (ii) it is applied a default aspiration criterium when the third alternative selects the "least" tabu move.

The cost function used on the iterative partition process considers as being optimum a partition alternative that respects the target architecture constraints and achieves the design requirements. To reach this goal, the function includes penalty terms, which measure the degree the partition alternative does not respect each constraint or requirement. The constraints apply to the area of the hardware partitions data path and to the area of the respective control unit, while the requirement applies to the system performance.

2.3 Metrics estimation

Like the majority of the approaches, the estimation operates over the system graph modelled with PSMfg. It uses an hardware model with data path and control unit and a software model that considers the code optimizations as a factor obtained by simulation. The implemented estimation methodology attempts to generate accurate estimates, while keeping the computation time as low as possible. To obtain accurate estimates, detailed models for the resources used on the implementations were developed, especially the hardware and communication models, and the estimation runs in two abstraction levels: system and program-state. The incremental update of the estimates and the estimation in two levels both help to decrease the computation time.

At the program-state abstraction level, low level estimates needed by the system level estimates are computed, the computations are performed once per partition session and the estimates are more accurate. At the program-state level, estimates for metrics relative to the system objects are computed. Examples of these metrics are the objects computation time, the area occupied by functional units, multiplexers and variables, the read/written variables and the program-states that read/write variables. At the system level, higher level estimates are obtained, the computations are repeated on every iteration of the partition process, the estimates are less accurate and, whenever it is possible, the estimates are updated instead of redo all the computations. The metrics estimated at the system level are the system performance and the area occupied by the data path and the control unit of the hardware partitions.

The developed hardware model is detailed and takes part on the computation of the system performance and the area occupied by the hardware partitions. According to this model, the area of a partition includes the area of the data path and the area of the control unit. The data path considers the functional units, the storage elements, the interconnection resources and the resources of the interface with other partitions. Apart from one constant, the area of the control unit equals the area of the state machine associated with the partition data path. The area of the state machine includes the state register, the output logic and the next state logic.

3 Target architecture

The target architecture presently supported by the partition methodology includes an EDgAR-2 reconfigurable platform and its host system. EDgAR-2 is built with FPGA/CPLD devices, interfaces the host system through PCI bus and it is fully in system programmable [1]. The architecture of the board is composed of an array of 4 couples (control unit, data path), called processor modules. The processor modules are interconnected in a linear way with dedicated buses, forming a pipeline, and they are also connected to a different byte from the 32 bits PCI data bus. Each processor module is implemented with a Xilinx 4013XL FPGA – the data path – and a XC95108 CPLD – the control unit.

4 Case studies

The partition methodology was validated with a detailed analysis of two case studies: the application of a Sobel filter to an image (convolution) and the DES[4] cryptography algorithm; the first one is oriented for a software implementation and the latter suggests an hardware implementation.

The application of a Sobel filter F (with X by Y pixels) to an image I, runs through two steps: (i) for

[4] **D**ata **E**ncryption **S**tandard.

every pixel (j,i) of the original image I, which colour is $I(j,i)$, an area with the filter size and centered on pixel (j,i) is convoluted with the filter F, generating a new value $If(j,i)$ for pixel (j,i) (equation 1); (ii) with the minimum and maximum of the filtered image If, $m(If)$ and $M(If)$ respectively, the filtered image is normalized to the colour range of the original image $(r(I))$, generating the filtered and normalized image In (equation 2).

$$If(j,i) = \sum_{k=0}^{Y-1} \sum_{l=0}^{X-1} I(j - \lfloor \frac{X}{2} \rfloor + l, i - \lfloor \frac{Y}{2} \rfloor + k) * F(l,k) \quad (1)$$

$$In(j,i) = \frac{r(I)}{M(If) - m(If)} * [\, If(j,i) - m(If) \,] \quad (2)$$

The implemented DES algorithm applies a set of transformations to the input data (sample), which depend on these data and on the secret key. This key is also altered during the different iterations of the encrypt process. Every sample to encrypt goes through an initial permutation IP, a set of transformations that depend on the secret key and a final permutation FP, inverse of IP (figure 4). The set of transformations that depend on the secret key is defined by an encryption function f and a key scheduling function KS.

The function f includes the expansion E, the substitution tables *S-box* and the permutation P. The information generated by the initial permutation IP is split in two 32 bits halves: the least significant part (R) feeds function f and the most significant part (L) is the input for an exclusive-OR operator. At the end of a round, the two halves of the sample to encrypt are swapped and the round is repeated. The algorithm evolves in 16 rounds, in order to "circulate" the sample to be encrypted.

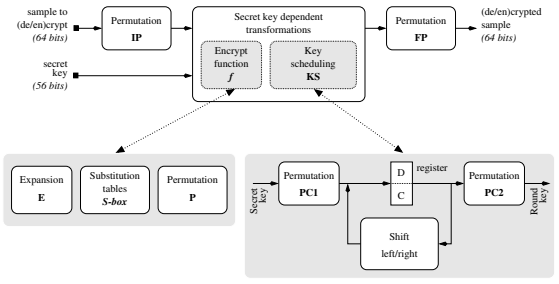

Figure 4: Block diagram of the DES algorithm.

The key scheduling KS generates a 48 bits key for each of the 16 rounds of the DES algorithm, through a linear combination of the 56 bits secret key. The KS module includes a permutation $PC1$, a register, a permutation $PC2$ and a shift left (right) operator, applied on the encrypt (decrypt) process.

The dimension of the partition problem associated with both examples and the parameters used on the resolution with the tabu search algorithm are synthesized on table 1. The high number of objects indicated for both examples is a consequence of using explicit parallelism at the system description.

Example dimension	***Convolution***	***Cryptography***
Nº partitions	5	5
Nº objects	217	372
Parameter		
Nº iterations	43400	74400
$nBest$	300	400
$pMoves$	20%	20%
$nRand$	4	4
TT_{move}	20	25
TT_{iMove}	18	22
TT_{obj}	15	20

nBest - Number of iterations since the best partition solution was found.
pMoves - Percentage of objects to be moved when the initial solution of a new search is created.
nRand - Number of searches without improving the best partition solution in order to execute "random" moves when creating the initial solution of the next search.
$\boldsymbol{TT_{move}}$ - Moves tabu tenure.
$\boldsymbol{TT_{iMove}}$ - Inverse moves tabu tenure.
$\boldsymbol{TT_{obj}}$ - Objects tabu tenure.

Table 1: Parameters used on the partition process with the tabu search algorithm.

5 Experimental results

The best partition solution, generated by tabu search for the DES example, assigns program-states and variables to partitions (SW or $HW1$ to $HW4$) as it is illustrated in figure 5. The objects in the upper part of the figure represent PSM variables and the remaining objects are the PSM program-states equivalents, for rounds 1 up to 6 and for round 16.

When the automatic partition solutions are compared with manually optimized hardware/software implementations, the measured performance of the best automatic partition solution reached 72 to 92% of the manual implementation performance, being superior on the cryptography example. The different experiments done with the mentioned examples always ended in feasible partition solutions, *e. g.*, solutions that respect the target architecture constraints.

The accuracy of the system performance estimates ranged from 82 to 98%, being higher on the cryptography example due to its lower complexity. A fidelity ranging from 83 to 100%, almost coincident with the accuracy range, suggests that the computed estimates are reliable. The accuracy of the estimates for the area occupied by the hardware partitions data path was 92 to 99%, being identical on both examples. The accuracy of the estimates for the area occupied by the hardware partitions control unit ranged from 89 to 96%,

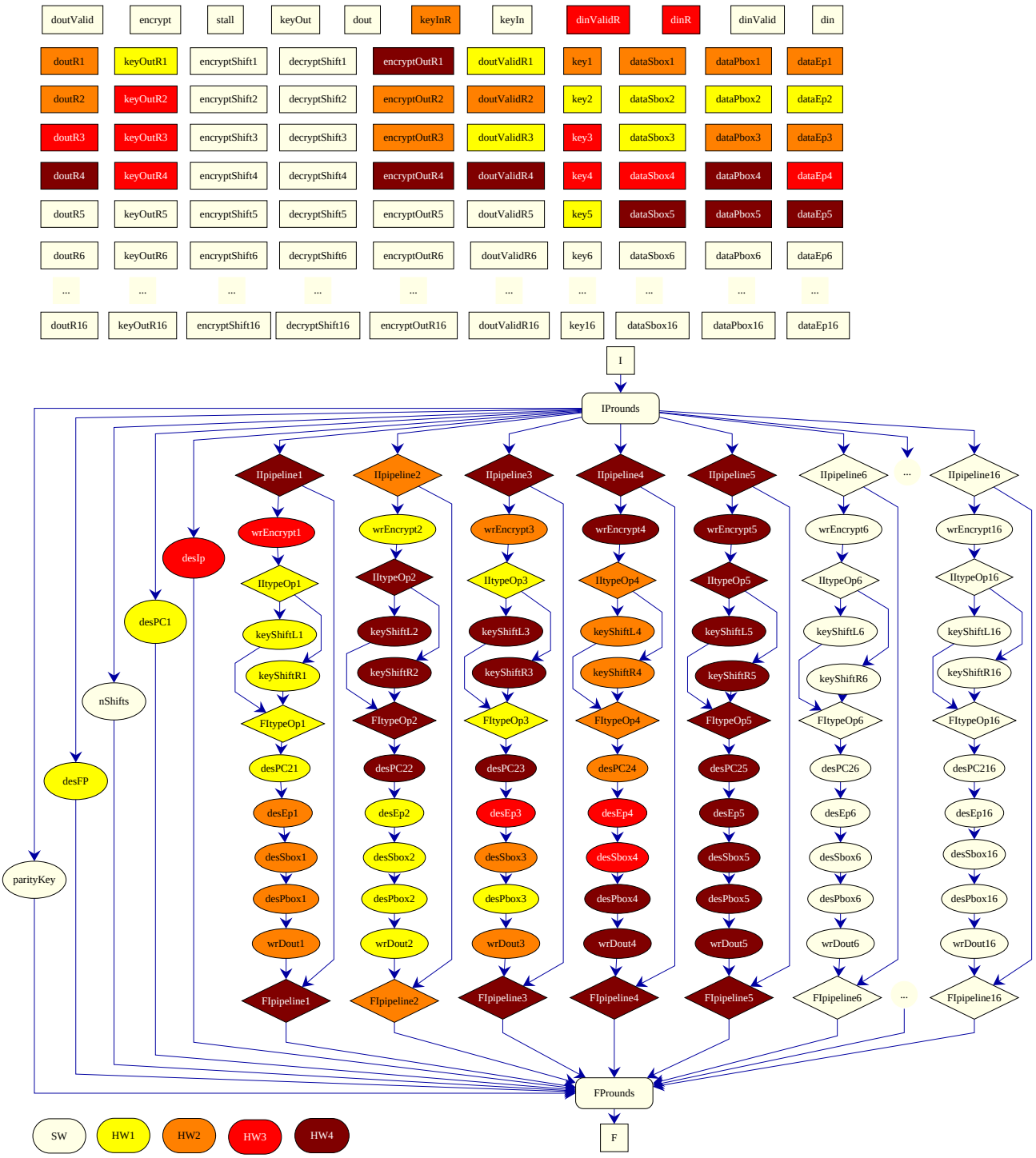

Figure 5: PSMfg model illustrating the best partition solution from the TS algorithm.

with very close results on both examples. The obtained results show that the control unit area depends mainly on the state register area, that in turn is proportional to the number of states. For the whole set of metrics and examples, the accuracy and fidelity of the estimates were always above 82%, a very rewarding result (table 2).

The time complexity $\mathcal{O}(nObj)$, expected for the tabu search algorithm, was experimentally proved. Since the computation time varies linearly with the number of objects on the system description, on large sized systems the time required to find the best partition solution is high. However, in the majority of cases, the first searches of the partition process generate a solution with quality.

The automatic synthesis of the interface between partitions is a straightforward implementation that uses the data from the estimation of the area occupied by the resources of the interface between partitions and the communication time between partitions.

Metric	*Convolution* (%)	*Cryptography* (%)
automatic vs manual solution performance	72	80-92
accuracy of performance estimates	82-83	97-98
fidelity of performance estimates	83	100
accuracy of areaHW(DP) estimates	98	92-99
accuracy of areaHW(CU) estimates	91	89-96

Table 2: Results obtained with the partition process.

6 Conclusions and future work

The obtained results show that the best automatic solution from the TS algorithm achieves 72 to 92% of the manual partition solution performance. This is an interesting result limited by (i) the optimizations introduced on the manual solution implementation, (ii) the simple software estimation model and (iii) the fine granularity used with the objects. The different experiments always ended on feasible partition solutions, which proves that the partition process is adequately controlled by the evaluation functions.

The accuracy of the performance estimates, the area of the data path and the area of the control unit estimates, was high. Measuring the data path (control unit) area with CLBs (products) leads to a better accuracy. The estimates accuracy obtained with both examples, DES and convolution, was very close. This consistence on the accuracy suggests a reliable estimation. For all metrics and examples, the accuracy and fidelity of the estimates was always above 82%, an interesting result that in many cases overcomes the published results.

The time complexity $\mathcal{O}(n)$, foreseen for the implemented TS algorithm, was confirmed on the experiments performed with *parTiTool*. The time necessary to compute the best partition solution is high, but in most cases 10% of this time is sufficient to find a solution that achieves a performance close to 90% of the best solution.

Some directions are being considered for future work: (i) replace the PSM by UML Statecharts in order to integrate the partition methodology on a broader embedded systems devellopment framework; (ii) evaluate the methodology with differentiated and more complex systems; (iii) implement other iterative algorithms – beyond TS and SA – where different optimization strategies may lead to better results with some examples, to increase the partition success; (iv) optimize the system performance estimation, to improve the performance of the partition methodology, strongly dependent on the time needed to estimate this metric.

References

[1] António Esteves. EDgAR-2: Highly Re-configurable Digital Emulator. Technical Report UMDITR9805, Dep. Informática, Universidade do Minho, Braga, Portugal, December 1998.

[2] Derrick Morris, Gareth Evans, Peter Green, and Colin Theaker. *Object Oriented Computer Systems Engineering*. Springer-Verlag, Applied Computing Series, 1996.

[3] Daniel Gajski, Frank Vahid, and Sanjiv Narayan. A System-Design Methodology: Executable-Specification Refinement. In *Proceedings of the European Conference on Design Automation*, 1994.

[4] Ralf Niemann. *Hardware/Software Co-design for Data Flow Dominated Embedded Systems*. Kluwer Academic Publishers. Boston, USA, 1998.

[5] G. Borriello, P. Chou, and R. Ortega. *Embedded System Co-design: Towards Portability and Rapid Integration*, pages 243–264. *Hardware/Software Codesign*, Ed. M. Sami e G. De Micheli. Kluwer Academic Publishers, Boston, USA, 1995.

[6] T. B. Ismail and A. A. Jerraya. Synthesis Steps and Design Models for Codesign. *IEEE Computer*, pages 44–52, 1995.

[7] K. Van Rompaey, D. Verkest, I. Bolsens, and H. De Man. CoWare - A Design Environment for Heterogeneous Hardware/Software Systems. In *Proc of the European Design Automation Conference (EURO-DAC)*, 1996.

[8] M. Chiodo, D. Engels, P. Giusto, H. Hsieh, A. Jurecska, L. Lavagno, K. Suzuki, and A. Sangiovanni-Vincentelli. A Case Study in Computer-Aided Codesign of Embedded Controllers. *Design Automation for Embedded Systems*, 1(1-2):51–67, 1996.

[9] António Esteves. *A Partition Methodology for Digital Embedded Systems Codesign* (in portuguese). PhD thesis, Dep. Informática, Universidade do Minho, Braga, Portugal, July 2001.

[10] António Esteves and Alberto Proença. A Partition Methodology to Develop Data Flow Dominated Embedded Systems. In *Proc of 1st Int Workshop on Model-Based Methodologies for Pervasive and Embedded Software (MOMPES), Hamilton, Canada*, 2004.

[11] Rajesh K. Gupta and Giovanni De Micheli. Constrained Software Generation for Hardware-Software Systems. In *Proceedings of the 3rd International Workshop on Hardware/Software Codesign*, pages 56–63. IEEE Computer Society Press, September 1994.

[12] M. Chiodo, P. Giusto, H. Hsieh, A. Jurecska, L. Lavagno, and A. Sangiovanni-Vincentelli. A Formal Methodology for Hardware/Software Co-design of Embedded Systems. *IEEE Micro*, August 1994.

[13] Petru Eles, Zebo Peng, and Alexa Doboli. VHDL System-Level Specification and Partitioning in a Hardware/Software Co-Synthesis Environment. In *Proceedings of the 3rd International Workshop on Hardware/Software Codesign*, pages 49–55. IEEE Computer Society Press, September 1994.

[14] Rolf Ernst, Jörg Henkel, and Thomas Benner. Hardware-Software Cosynthesis for Microcontrollers.

IEEE Design & Test of Computers, 10(4):64–75, December 1993.

[15] Kurt Mehlhorn and Stefan Näher. *The LEDA Platform of Combinatorial and Geometric Computing.* Cambridge University Press, 1999.

[16] Petru Eles, Krkrysztof Kuchcinski, and Zebo Peng. *System Synthesis with VHDL.* Kluwer Academic Publishers, 1998.

[17] Fred Glover and Manuel Laguna. *Tabu Search*, pages 70–150. *Modern Heuristic Techniques for Combinatorial Problems*, Ed. Colin Reeves. McGraw-Hill, 1995.