A Partition Methodology to Develop Data Flow Dominated Embedded Systems

António J. Esteves and Alberto J. Proença
Department of Informatics, University of Minho
Braga, Portugal
{esteves,aproenca}@di.uminho.pt

Abstract

This paper proposes an automatic partition methodology oriented to develop data flow dominated embedded systems. The target architecture is CPU-based with reconfigurable devices on attached board(s), which closely matches the PSM meta-model applied to system modelling. A PSM flow graph was developed to represent the system during the partitioning process. The partitioning task applies known optimization algorithms - tabu search and cluster growth algorithms - which were enriched with new elements to reduce computation time and to achieve higher quality partition solutions. These include the closeness function that guides cluster growth algorithm, which dynamically adapts to the type of object and partition under analysis. The methodology was applied to two case studies, and some evaluation results are presented.

Keywords: partitioning, hardware/software co-design, PSM meta-model, tabu search, cluster growth

1 Introduction

This paper describes an automatic partition methodology oriented to develop data flow dominated, medium complexity and real time embedded systems, where a processing element coupled to FPGA/CPLD board(s) [1] form a reconfigurable architecture [2]. Since this architecture includes hardware and software components, the present work applies the hardware/software codesign paradigm.

Partitioning is an NP-complete optimization problem that assigns system objects to the target architecture components and defines its startup time (scheduling), to achieve the designer objectives quantified by a cost function [3]. The partition process converts an unified and uncommitted system representation to a multi-part representation committed to the target architecture components. The present approach performs a functional, inter-component and automatic partition.

The partition task is part of a development methodology that covers all phases of systems development [4]. It is based on an operational approach [5], it runs

at a high abstraction level and it takes advantage of the object oriented modelling paradigm to reduce complexity and design time. Common to object oriented approaches, it uses multi-view modelling to describe the objects, the dynamic and the functional perspectives of systems. Following an operational approach, an executable specification is developed, which runs through a set of refinements and transformations to achieve a system implementation. When compared to the methodology followed by the MOOSE approach [6], the proposed methodology has some advantages: (i) the state transition diagrams (STD) are replaced by PSM[1] models [7], 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, a prototype tool was implemented, *parTiTool*, and its capabilities were compared to other approaches, following the structure introduced in [8]. 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 (control, data or data+control), the type of validation (simulation or co-verification) and the modelling style (homogeneous or heterogeneous). Figure 1 shows where *parTiTool* fits in the graph and how it relates to other approaches. Most approaches adopt homogeneous modelling style, where the only allowed validation method is simulation. Systems are described with a software oriented language (C, a C variant, Occam or C++) or an hardware oriented language (VHDL, Verilog or HardwareC). 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.

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 [9], Cosmos [10], CoWare [11] or Polis [12]) does not execute partition automatically. None of the approaches completely supports the automatic partition and the synthesis of 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. Current *parTiTool* prototype implementation does not support yet more than one microprocessor, due to the target evaluation architecture. However, a R&D track is being prepared to merge this project with current adaptive load and data scheduling in parallel and distributed systems [13].

The paper is organized in 4 sections. Section 2 describes the proposed partition methodology, namely the formal description of the partition process, the approach followed to model the system and its internal representation, the construction and

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

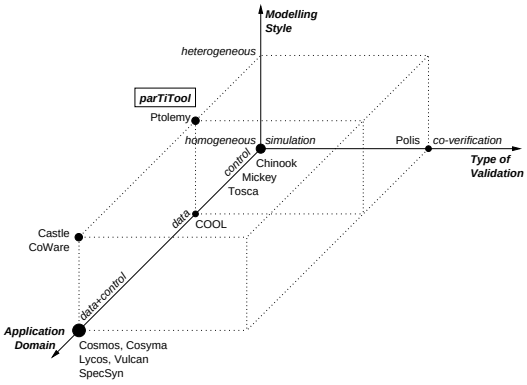

Figure 1: Categorization of approaches by modelling support.

improvement of partition solutions with cluster growth and tabu search algorithms and the metrics estimation required by the evaluation functions. Section 3 presents the prototype system used to validate the partition methodology validation – target architecture and applied tool – and summarizes the case studies and the obtained results. Section 4 closes with conclusion remarks and directions for future work.

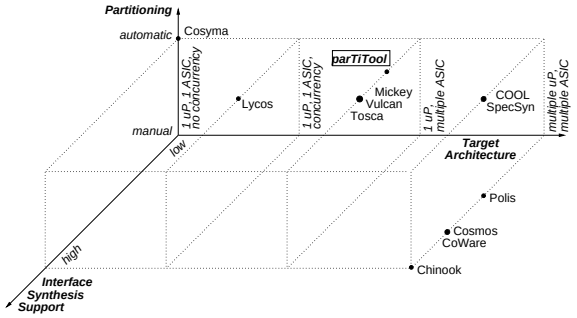

Figure 2: Categorization of approaches by implementation support.

2 Partition methodology

The presentation of the partition methodology starts with the formal description of the partition process. Given an unified representation for the system (see below, in *system modelling*), the partition process generates a description for each component of the target architecture to be used on the implementation of the system. To reach this goal, the set of objects on system description must be divided into a series of disjunct sub-sets that will be assigned to the different components of the target architecture. The task that divides the set of objects on sub-sets is guided by the target architecture constraints and the design requirements. In the present work, the objects represent program-states or variables from the system PSM model. A

formal definition of the partition process follows below.

Given the set of objects $O = \{o_1, o_2, ..., o_n\}$ that models the system functionality, the set of constraints $Cons = \{c_1, c_2, ..., c_m\}$ and the set of requirements $Req = \{r_1, r_2, ..., r_p\}$ that define the feasibility and the quality of the partition alternatives to be generated, the partition process generates several sub-sets (or partitions) $H_1, ..., H_{nh}, S_1, ..., S_{ns}$, where $H_i \subseteq O$, $S_i \subseteq O$, $\{H_i\}_{i=1..nh} \cup \{S_j\}_{j=1..ns} = O$, $H_i \cap S_j = \oslash$, $H_i \cap H_k = \oslash$ (with $k = 1..nh$ and $k \neq i$) and $S_j \cap S_l = \oslash$ (with $l = 1..ns$ and $l \neq j$).

The selection of a partition solution, among all that were analyzed by the partition algorithm, implies a cost function F_{cost}. This function uses the sub-sets of objects assigned to hardware $H = \{H_1, ..., H_{nh}\}$, the sub-sets of objects assigned to software $S = \{S_1, ..., S_{ns}\}$, the set of constraints $Cons$ and the set of requirements Req to return a value that measures the solution quality. The iterative partition algorithm is defined by the function

$$PartAlg(H, S, Cons, Req, F_{cost}()) \tag{1}$$

returning H' and S' that verifies

$$F_{cost}(H', S', Cons, Req) \leq F_{cost}(H, S, Cons, Req) \tag{2}$$

when the applied cost function returns the minimum value under the best partition solution circumstances.

The value generated by the cost function is obtained from estimated metrics, related to the system constraints and requirements.

To execute the partial tasks needed by the partition process, the modules identified in figure 3 were used. Beyond the module that performs the conversion between the models used externally and internally by the partition process, the developed partition methodology includes partition algorithms (constructive and iterative), evaluation functions (closeness and cost) and metrics estimators. The following sections describe the modelling that is relevant for partition and the partition itself, with emphasis on the applied algorithms and evaluations functions and briefly presenting the metrics estimation.

2.1 System modelling

In related approaches, the uncommitted systems are commonly modelled with meta-models such as CDFG [14] [15], DFG [16], FSM [17], Petri net [18], CSP [19], an extended version of a previous meta-model [20], or a combination of these meta-models [6]. In spite of the meta-model diversity, most approaches transform the uncommitted system model into a flow diagram representation. The type of objects handled during the partition process is constrained by the selected meta-model, and the several approaches may present quite a different granularity, as a consequence of using distinct meta-models.

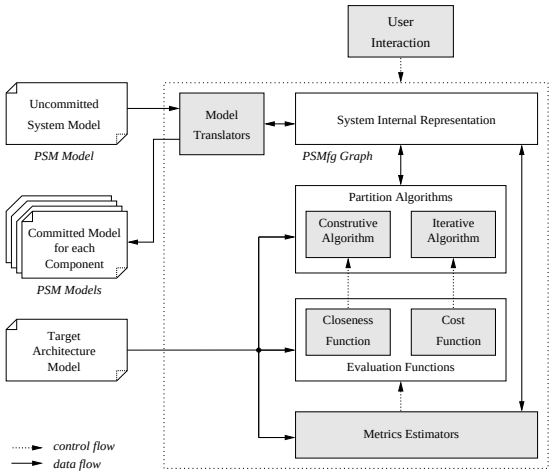

Figure 3: The modules of the partition methodology.

The PSM meta-model was selected to describe the systems at the partition process interface, which combines an HLL/HDL meta-model with HCFSM[2] [7]. PSM adequately supports complex embedded systems modelling, since it includes the best features from both meta-models: behavioural hierarchy, concurrency, state definition, support to handle algorithmic and data complexity, behaviour completion, possibility of including exception handling and a graphical representation. Besides, modelling with PSM is a very intuitive task. The strongest limitations of PSM are the lack of structural hierarchy and automatic support to formally validate the models. In the present approach, the VHDL language was selected to describe variables and leaf program-states. VHDL allows an explicit and elegant modelling of communication and synchronization among concurrent activities.

A PSM model is described by an 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 chosen programming language. If the program-substates are concurrent they are all active at the same time; if they are sequential, just one program-substate can be active at a certain time.

On a composite program-state, the order by which sequential program-substates get active is determined by the directed arcs connecting them. There are two type of directed arcs: arcs that represent a transition when the substate activity is terminated and simultaneously the condition associated with the arc becomes true, and arcs that represent a transition immediately after the condition associated with the arc becomes true. A transition on a directed arc means that the target substate will

[2] **H**ierarchical **C**oncurrent **F**inite **S**tate **M**achine.

become active.

To represent a PSM model textual and graphic notations can be used.

Internal representation

To describe systems during the partition process a CFG type meta-model was developed: the PSM flow graph or simply PSMfg. The most relevant requirement of the internal representation, not included on the PSM meta-model requirement list, is the possibility of associating the information generated during the partition process with the system model objects.

The motivations that lead to the development of a new meta-model were the need to automate the partition process and the availability of a library with graphic and computational support to edit graphs - LEDA[3] [21]. By means of a set of adaptations applied to the editor of generic directed graphs and the associated data structure, it was possible to obtain the computational support to operate on PSMfg graphs. The goals to achieve with the performed adaptations were: (i) to customize the graphic characteristics of the nodes, generating the set of node types that will be presented ahead; (ii) to increase the nodes and edges functionality, in agreement with its type; and (iii) to introduce constraints on the interconnection between the different types of node.

A PSMfg model is an acyclic, directed and polar graph, represented by a $G = \{V, E\}$ data structure that includes the list of nodes V and the list of edges E. The graph is acyclic when no paths on the graph are closed, it is directed because each edge has a single direction and it is polar because it includes two nodes, one to enter and the other to exit from the graph, from which all other nodes are successors and predecessors, respectively [22].

The meta-model of the PSMfg represents the semantic of the PSM meta-model and all the information needed by the partition process, such as the metrics estimate and the assignment of objects to partitions. To control the granularity of the objects handled during the partition process, the PSMfg graph must be able to represent the program-states structure. Since the program-states functionality is described with VHDL, the PSMfg graph supports the following constructs of the VHDL language: the parallelism associated with processes, the conditional constructs (`if ... elsif` and `case`), the cycles (`while` and `for`) and the constructs that suspend processes (`wait`).

The nodes of a PSMfg graph represent the variables and the program-states of a PSM model, with the same or a thinner granularity, and they have associated with them information that is relevant to the partition process, namely:

- ◇ which partition the graph node was assigned;
- ◇ which partitions the designer establish as being forbidden for this node;

[3] **L**ibrary of **E**fficient **D**ata types and **A**lgorithms.

- ◇ the required information to estimate the area occupied by the hardware and the system performance, which refers to metrics like the functional units, the storage elements or the interconnection elements area, the variables read/written by the program-state associated with the node, the computation time, the time spend on communication with others program-states or the execution frequency.

The different node types the PSMfg meta-model uses are those that: (i) define the entry/exit point of the system graph; (ii) indicate where the (parallel) processes begin/end; (iii) define the begin/end of a conditional construct; (iv) represent the control part of a cycle; (v) force a waiting cycle; (vi) assert one or more signals necessary to a waiting cycle; (vii) represent a variable; and (viii) do not fit in any of the previous types.

The edges, representing the control flow between nodes, have associated a branch probability (relative to the source node) and a label.

2.2 System Partitioning

In the present work, partitioning is a two-step process: (i) compute an initial partition solution with a constructive algorithm and (ii) successively improve it with an iterative partition algorithm.

A constructive algorithm

The analysis of several constructive partition algorithms revealed that: (i) the application of an exhaustive algorithm is not feasible since it demands an unacceptable computation time; (ii) the cluster growth and hierarchical clustering algorithms create the partition solutions in distinct ways, but produce identical results; (iii) the ILP[4] methods generate *optima* solutions, do not require the application of an iterative optimization algorithm, but they demand a very high computation time and its formulation is hard to achieve; and (iv) PACE [23] and GCLP [3] algorithms, being strongly specific, are not attractive to be adapted to the present work. Since the solutions generated by the constructive algorithm feed the iterative improving process, its quality can be kept in a lower value. Thus, it was selected a constructive algorithm with a light implementation, the cluster growth (CG) algorithm. Although the optimization heuristic of the CG algorithm is quite simple, the capacity to generate solutions with quality is determined by the selected closeness function.

The process of creating a solution begins with the selection of the seed object for each partition. To select the partitions seed object, 4 methods were implemented: (i) random selection, (ii) manual assignment, (iii) combination of random selection with manual assignment and (iv) selection based on the communication among partitions. The manual assignment can be used to avoid that objects are assigned to an implementation for which they are clearly bad candidates. Having

[4] **I**nteger **L**inear **P**rogramming.

selected the seed object for each partition, the cluster growth algorithm assign the remaining objects to the best possible partition. The best partition is chosen by the closeness function defined in equation 5 of section 2.3.

An iterative algorithm

Simulated annealing [20] [18] [24] [25] is among the most commonly used iterative partition algorithms, but it is also frequent to use genetic evolution [8], implementations of the Kernighan/Lin algorithm [26] [27], tabu search [24] [25] and specific algorithms. The evaluation of these algorithms has shown that Kernighan/Lin algorithm has a limited capacity to avoid local minimum of the cost function, the simulated annealing algorithm presents a stronger potential than greedy and Kernighan/Lin algorithms to achieve *optima* solutions, but the computation time is very high, and the tabu search algorithm decreases the computation time bounding the search for partition solutions to the neighbourhood of these solutions. The genetic evolution algorithms reduce the design space more efficiently, but the capacity of convergence to the optimum partition solution is inferior. Having in mind that the primary goal of partitioning is to find partition solutions with quality, tabu search and simulated annealing were selected for the iterative process. A thorough study was carried out with tabu search algorithm, and the results are presented in this paper.

The tabu search method (TS) can be seen as an extension of the local search strategies, where a new solution is found on the neighbourhood of the present solution, applying a well defined set of rules [28] [29]. When the iteration n of the search process tries to minimize the cost function $F_{cost}(P_n)$, the new solution P_{n+1} is selected from the neighbourhood $V(P_n)$ of the present solution, applying an optimization criterium. In general, the criterium expresses the objective of selecting the best solution present on the neighbourhood. The neighbourhood of solution P_n can be defined by the set of all the alternatives that result from the application of a rule that modifies the characteristics or attributes of P_n. On the hardware/software partition problem, the transition from the present solution to a solution on its neighbourhood occurs when at least one object is moved from its current partition to a target partition, ending in a new solution. It is frequent an hardware/software partition problem to evaluate a high number of partition alternatives, which means that to find a solution with quality it is necessary a computation time equally high. To avoid that all the alternatives present on the current solution neighbourhood are evaluated, it is implemented a list with candidate solutions; this way, only a partial neighbourhood of the current solution is evaluated.

Although the tabu search is a local search strategy that tries not to stop in local minima of F_{cost}, its policy embodies other features. This strategy was named tabu search since in every iteration parts of the design space are forbidden, *e. g.*, some solutions are considered tabu. To reach this goal, the tabu search implements a flexible memory structure that supports several search strategies, like avoiding local minima. The flexible memory includes short term (STM), long term (LTM)

and medium term components (MTM). The short term components are based on the history of most recently visited solutions, the long term components are based on the most frequent solutions and the medium term components are oriented to solutions with quality and influent solutions. Using this information it is possible (i) to diversify the search, in order to escape the local minima, (ii) to intensify the search, to reinforce the convergence for the absolute minimum and (iii) to avoid cycles during the search.

To avoid a cycle during the search, the last L visited solutions are saved on the tabu list. While a solution P_n is on the tabu list, it is forbidden. This way, the search will not return, at least during L iterations, to a visited solution. The size of the tabu list, or the tabu tenure, is determinant to the evolution of a search, since it influences the restrictions applied to the design space that can be searched. For this reason, the more restrictive is a tabu, the less must be its tenure. The performed experiments resulted in the following recommendation: the objects and moves tabu tenure must be 5 to 10% of the number of objects on the system description.

The temporary exclusion of solutions does not result solely in advantages for the tabu search method. The disadvantages arise when high quality solutions, the goal of searching, are excluded from the search. To overtake the inconvenient caused by the high quality solutions exclusion, TS methods have a mechanism that allows to withdraw the tabu classification of a solution, assuming it may be a solution with quality. This mechanism is called aspiration criterium. They can be defined aspiration criteria by objective, by direction of search and by influence [28].

The tabu search algorithm iteratively tries to improve the provided partition solution, assembles all the components that participate on the search and controls its evolution. The implemented algorithm [2] is a modified version of the one discussed in [29]. Namely, it only searches a partial neighbourhood of the present solution, it has a richer set of evolution strategies to apply when there are no eligible solutions with quality, and it applies a more efficient improvement when none of the moves improves the cost of the present solution. Partial neighbourhood searching, decreases the computation time per iteration by a factor close to the number of partitions, but the design space exploitation is less complete. Since the partial neighbourhood contains the best solutions, it is introduced an intensification element on the search.

The tabu search algorithm runs until a predefined number of iterations is reached and each search runs while a predefined number of iterations without improving the best solution is not exceeded. The number of iterations that can be executed without improving the partition solution should not be neither too high - to avoid wasting iterations around a local minimum - nor too low - to increase the possibility of converging to a local (or absolute) minimum of the cost function.

On every iteration of the search process, the partial neighbourhood of the present solution is analysed and the move to be executed can be selected from one of the following ordered alternatives:

(1) The move that generates the largest improvement on the partition solution cost and that obeys one of the following conditions: it is not tabu or it is tabu but can be executed due to an aspiration criterium;

(2) The move that is not tabu and leads to the smallest increase on the partition solution cost; the cost of the solutions that result from the moves is decreased by the application of a negative improvement;

(3) The "least" tabu move, the least frequent move, the move that in the past resulted in the best cost variation or the move of the object that stays longer in the same partition.

At the end of every iteration the performed move, the inverse move and the moved object are classified as being tabu, the history is updated with the information about the move and the moved object, the moves and objects tabu tenure, the best solution found and the number of iterations are updated and, at the end of a search, a new initial solution is generated.

Experimental results show that, at the beginning of each search, the information saved on the history of moves and moved objects must be reset. Otherwise, the capacity of converge to the optimum solution is reduced, since the improvement used on the second move alternative, proportional to the number of iterations an object is not moved, would regularly select every object.

Applying all types of tabu classification can be very restrictive to the search. A subset of tabu classifications was selected, which decreases the dimension of the neighbourhood to be searched, helps to avoid cycles and does not place excessive restrictions to the search. The following tabu classifications were selected: (i) move a given object from a source partition to a target partition; (ii) all moves of a given object; and (iii) the inverse (move) of the move that originated the present solution.

The implemented TS method includes two types of aspiration criterium: (i) by objective, when the first alternative selects a move with quality that is classified as being tabu; and (ii) a default criterium, when the third alternative selects the "least" tabu move.

The implemented memory structure registers the history of performed moves and the history of moved objects. For each performed move, the history of moves saves the source and target partitions, the tabu tenure (STM), the execution frequency (LTM) and the achieved cost variation (MTM); for each moved object, it saves the tabu tenure (STM), the frequency of move (LTM), the number of iterations an object remains on the same partition (LTM) and the achieved cost variation (MTM).

It was implemented a neighbourhood with a simple structure since a neighbourhood with a complex structure would increase greatly the computation time. While on a partition problem with $nObj$ objects and $nPart$ partitions, the size of the simple neighbourhood is $nObj * (nPart - 1)$, on a generic complex neighbourhood, where each iteration executes a series of $nMoves$ moves, the number of alternatives that make up the neighbourhood is defined by the equation 3. The

value defined by this equations is much higher the number of alternatives on a simple neighbourhood. A complex neighbourhood favours the diversification on the search, which means an increased capacity to avoid the local minimum but also an increased difficulty to converge to the optimum partition solution.

$$size(V) = (nPart - 1)^{nMoves} * \mathcal{C}_{nMoves}^{nObj} = (nPart - 1)^{nMoves} * \frac{nObj!}{nMoves! * (nObj - nMoves)!} \quad (3)$$

The partial neighbourhood, or the list of candidate solutions, considered on every iteration of the TS algorithm is a subset of the present solution neighbourhood, with a size that remains fixed during all the search process and equals the number of system objects. The $(nPart - 1)$ moves per object that define the neighbourhood were decreased to only one move per object on the partial neighbourhood, decreasing the computation time by a factor close to the number of partitions. The subset of moves that define the partial neighbourhood is made up by the best move for each object of the system description. The best moves are computed by a function identical to the closeness function of the cluster growth algorithm (F_{prox} on equation 5).

Part of the TS algorithm potential is consequence of executing several searches, each one with a different initial solution. The method used to generate the initial solution of the searches combines two strategies: intensification - the new initial solution results from the best evaluated partition solution - and diversification - the assignment of a significant percentage of objects is modified, according to the long term memory. The rule is to execute the least frequent moves, but after a number of searches without improving the best solution, the choice can be to move the least frequently moved objects to a randomly selected partition. The random selection reinforces the diversification on the search. Given that the percentage of moved objects is a parameter of the algorithm, it is possible to control the relation between the intensification and the diversification applied on the generation of a new initial solution. The suggested value for the percentage of objects to be moved is 20%.

The implemented algorithm includes the following intensification elements:

- ◇ to create the list of candidate solutions with the highest quality solutions present on the current solution neighbourhood;
- ◇ to create the initial solution for a new search based on the best evaluated solution;
- ◇ to select, for third evolution alternative of the TS algorithm, the move that in past resulted in the best cost variation;

and the following diversification elements:

◇ to apply, on the second evolution alternative of the TS algorithm, a cost improvement based on the number of iterations the objects remained in the last partition P_k they were assigned ($NIMP_k$); this improvement, described by equation 4, strongly favours the move of the objects that are not moved regularly, since they have a high $NIMP$; thus, the search is directed to less explored zones and a diversification component is introduced on the search;

$$improvement(P_k) = -\frac{NIMP_k}{nObj} \tag{4}$$

◇ to create the initial solution of a new search moving a percentage of the least frequently moved objects or a percentage of objects selected randomly (after a number of searches);

◇ to select, as the third evolution alternative of the TS algorithm, the least frequent move or move the object that remains longer on the same partition.

2.3 Evaluation functions

This section describes the evaluation functions (closeness and cost) that guide the partition algorithms (constructive and iterative) on the creation and improvement of partition solutions.

Closeness function

The best partition used to assign the objects, on every iteration of the constructive partition process, is chosen by the closeness function defined in equation 5.

$$F_{prox} = f \left[\begin{array}{c} F_{var}(M_{com1}) \\ F_{psHwSw}(M_{cmp1}, M_{cmp2}, M_{com2}) \\ F_{psHw}(M_{area}, M_{com2}) \end{array} \right] \tag{5}$$

where M_{com1} (M_{com2}) represents the communication intensity among a variable (program-state) and the program-states (variables) assigned to the partition, M_{cmp1} (M_{cmp2}) is the software (hardware) computation time of a program-state and M_{area} is the area occupied by all the variables and program-states assigned to the partition. The F_{var} function is used on variables assignment and the F_{psHwSw} and F_{psHw} functions are used on the program-states assignment. On every moment of the constructive process, the F_{prox} function measures the closeness among the object to be assigned and the objects previously assigned to each partition.

If the object to be assigned is a variable that is a bad candidate to hardware, meaning that the area it occupies in hardware exceeds a defined limit, the F_{var} function suggests an assignement to software. If the variable is not a bad candidate to hardware, it is assigned to the partition that presents the higher communication intensity with this variable, *e.g.*, to the partition p that presents the best $M_{com1}[p]$ value.

When a program-state is being assigned, if the F_{psHwSw} function indicates that software is the best partition to assign it, the program-state is immediately assigned to software. Otherwise the best hardware partition is selected by F_{psHw}, a function that is more appropriated to distinguish the assignment to the different hardware partitions.

For example, the metric M_{com1} used to select the best partition p to assign a variable v, is computed with equation 6. The communication intensity $M_{com1}[p]$ simply measures the number of times the variable v is read/written by the program-states assigned to the partition p.

$$M_{com1}[p] = \sum_{o \in (rdO(v) \cap p)} rdV(o).nRd(v) * rdV(o).pRd(v) * FN(o) + \sum_{o \in (wrO(v) \cap p)} wrV(o).nWr(v) * wrV(o).pWr(v) * FN(o) \quad (6)$$

where

- $rdO(v)$ ($wrO(v)$) is the set of program-states that read (write) the variable v;
- $rdV(o)$ ($wrV(o)$) represents the set of variables read (written) by the program-state o;
- $rdV(o).nRd(v)$ ($wrV(o).nWr(v)$) is the number of times the variable v is read (written) by o on every execution;
- $rdV(o).pRd(v)$ ($wrV(o).pWr(v)$) is the probability of variable v to be read (written) by o;
- $FN(o)$ is the execution frequency of o.

Cost function

The cost function applied on the iterative partition process considers as being optimum a partition solution that respects the target architecture constraints and achieves the design requirements, as opposed to considering as being optimum a solution that uses the least hardware area and/or achieves the best performance. To reach this goal the function includes a term, per constraint or requirement, whose value is proportional to the degree this constraint or requirement is not observed on the partition alternative (equation 7).

$$F_{cost}(H, S, Cons, Req) = \sum_{i=1}^{3} K_i * f_i(M_i, C_i) \quad (7)$$

where

- H (S) is the set of hardware (software) partitions;

- $\diamond$ $Cons = \{C_1, C_2\}$ is the set of design constraints, with C_1 being the constraint applied to the area of the hardware partitions data path (M_1) and C_2 the constraint applied to the area of the respective control unit (M_2);
- $\diamond$ $Req = C_3$ is the performance required from the system (M_3);
- $\diamond$ M is the set of metrics M_i, whose constraints $Cons$ and requirement Req apply to;
- $\diamond$ K_i is the coefficient applied to the metric M_i;
- $\diamond$ $f_i(M_i, C_i)$ represents the contribution of the metric M_i to the cost function and it is defined by equation 8.

$$f_i(M_i, C_i) = \begin{cases} \sum_{P_j \in H} MAX\ [excess(M_i[P_j], C_i)\ ,\ 0] & ,\ i=1,2 \\ MAX\ [excess(M_3, C_3)\ ,\ 0] & ,\ i=3 \end{cases} \quad (8)$$

where

- $\diamond$ $M_i[P_j]$ is the value of metric M_i for the hardware partition P_j;
- $\diamond$ $C_i[P_j]$, the value of the design constraint C_i applied to the hardware partition P_j, was replaced by C_i on equation 8; on the considered target architecture, the pairs (FPGA,CPLD) that implement the pair (DP,CU) of the hardware partitions include the same devices;
- $\diamond$ the term $excess(m, c)$ is given by

$$excess(m, c) = \frac{m - c}{c} \quad (9)$$

The estimates for the area (M_1 and M_2) are computed by partition, while the estimate for performance (M_3) is relative to all the system.

2.4 Metrics estimation

Metrics estimation aims to compare partition alternatives, which requires a high degree of fidelity rather than a high accuracy. However, it is expected that a high accuracy corresponds to an equally high degree of fidelity.

The estimation operates on the system graph, modelled with PSMfg, considers an hardware model (with data path and control unit), a software model (with a predefined set of instruction types) and a communication model (for inter-partition communications). The code optimization performed by the compiler - related to pipelining, superscalarity and memory hierarchy - is measured as a factor obtained by simulation. This procedure is acceptable on most partition problems applied to

embedded systems. One difference to a significant part of the approaches, is the emphasis given to the estimation of metrics related to inter-partitions communications.

To obtain accurate estimates, detailed models for the used resources were developed, especially the hardware and communication models, and the estimation runs in two abstraction levels: program-state and system. The incremental update of the estimates and the estimation in two levels both help to decrease the computation time.

Low level estimates, which are used by the system level estimates, are computed at the program-state abstraction level, the computations are performed once per partition session and the estimates are more accurate. Estimates for metrics relative to the system objects are computed at the program-state level. Examples of these metrics are the software and the hardware computation times, the area occupied by functional units, multiplexers and variables, the read/written variables and the program-states that read/write variables. To obtain these estimates, low level metrics are required: these include the execution time of the arithmetic/logic operators and the area occupied by multiplexers, arithmetic/logic operators and memory elements.

At the system level, metrics are estimated at a higher level and the computations are repeated on every iteration of the partition process. The estimates are less accurate and, whenever possible, the estimates are simply updated. 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. System performance is computed through explicit scheduling at the state-program level. By ignoring the scheduling at the system level, the computation time is decreased and the obtained performance tend to be over estimated.

The computation of the execution times follows a simple software model, that estimates computation time by type of executed instruction (the built prototype follows the IA-32 architecture model) and considers the optimizations performed by the compilers as a factor obtained by simulation.

The developed hardware model focus on the area of a partition, which includes the area of the data path - the functional units, the storage elements, the interconnection resources and the resources of the interface with other partitions - and the area of the control unit - the area of the state machine associated with the partition data path, which includes the state register, the output logic and the next state logic. Experimental results confirmed the state register as the dominant term on the area of the state machine, ranging from 60 to 80%.

The developed communication model defines the timings and the resources associated with the communication between partitions. The model supports the register access communication mechanism, by polling and by interruption. At the beginning of every search of the iterative partition process, estimates for communication times are computed. These estimates will be updated whenever an object is moved from one partition to another one, but only for the moved object and/or those objects that communicate with the moved objects.

3 Validation of the partition methodology

The proposed methodology was validated on a CPU-based architecture coupled to a reconfigurable board (briefly described below), through two case studies that represent data flow dominated embedded systems: one clearly suggesting a software implementation, while the other is oriented for an hardware implementation.

A quantitative evaluation compared automatically generated solutions with manually optimised hardware/software implementations, looking into two main results: the quality of the partition solutions (measured by feasibility and performance) and the quality of the estimates (measured by accuracy/fidelity). The methodology can be further evaluated by its performance, *e. g.*, by the computation time needed to generate the partition solutions and the support to implement these solutions, namely to synthesize the interface between partitions.

3.1 Prototype system

The prototype system applied on the partition methodology validation includes a target architecture and a partitioning tool.

The considered target architecture contains a reconfigurable platform (EDgAR-2) and its host system. The EDgAR-2 board is an FPGA/CPLD based system, with a PCI interface and fully in system programmable (ISP) [1] [30]. The board structure, shown in figure 4, contains an array of 4 pairs (control unit, data path), called processor modules (PMs), which are 2-way interconnected with dedicated buses, forming a PM pipeline; they are also connected to a different set of 8 lines in the 32 bits PCI data bus. FPGAs implement the data paths, while CPLDs are better suited to implement the control units.

The EDgAR-2 architecture was designed to directly accommodate a *finite state machine with data path* (FSMD) model. Since the architecture implements several concurrent FSMDs, it is suitable to map descriptions modelled with *concurrent FSMDs* (CFSMD) [31], *hierarchical concurrent FSMD* (HCFSMD) or *program state machine* (PSM) meta-models [32].

Although EDgAR-2 may not be considered a typical reconfigurable board - it is composed of both FPGAs and CPLDs, and it lacks on board RAM - it is particularly adequate to validate a general purpose hardware/software partition methodology due to these extra challenges.

The applied tool was $parTiTool$, a framework based on the LEDA library [21] and which allows the visualisation, edition and partitioning of PSMfg graphs. This framework also includes support to detect errors on the graphs structure and to visualise the output of the partitioning process. The major part of the operations needed by the graphs visualisation and edition is supported by the classes `GRAPH` and `GraphWin` of the LEDA library.

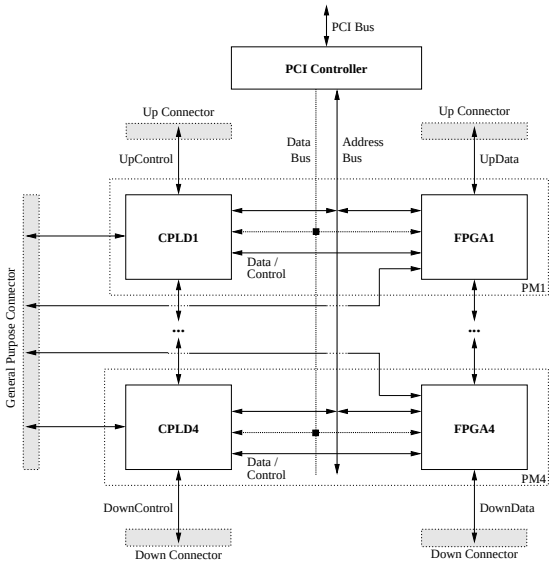

Figure 4: The EDgAR-2 platform architecture.

3.2 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[5] cryptography algorithm [33]; 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 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 10); (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 11).

$$If(j,i) = \sum_{k=0}^{Y-1} \sum_{l=0}^{X-1} I(j - \lfloor \frac{X}{2} \rfloor + l, i - \lfloor \frac{Y}{2} \rfloor + k) * F(l,k) \tag{10}$$

$$In(j,i) \; = \; \frac{r(I)}{M(If) - m(If)} * [\, If(j,i) - m(If) \,] \tag{11}$$

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

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

also altered during the different iterations of the encrypt process. Every sample to encrypt goes trough an initial permutation IP, a set of transformations that depend on the secret key and a final permutation FP, inverse of IP (figure 5). 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 splitted 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.

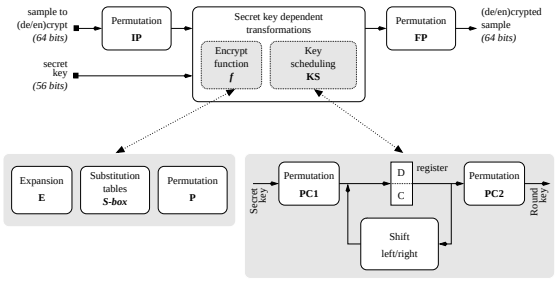

Figure 5: Block diagram of the DES algorithm.

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.

3.3 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 6. The objects in the upper part of the figure represent PSM variables and the remaining objects are the PSM program-states equivalents.

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. These results can be improved by detailing the estimation models and by tuning the granularity of system model objects, which will significantly increase the computation time. The different ex-

Example dimension	***Convolution***	***Cryptography***
N^o partitions	5	5
N^o objects	217	372
Parameter		
N^o 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.

periments done with the mentioned examples always ended in feasible partition solutions, *e. g.*, solutions that respect the target architecture constraints, a proof that the applied closeness and cost functions correctly control the partition process.

The accuracy and fidelity of the estimates for the performance and for the area occupied in hardware were also evaluated. 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%, 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. The results obtained with the partition process are summarised on table 2.

When it was decided to compute accurate estimates, the performance of the partition methodology tool ended penalized. One way of improving the tool performance is to optimize the estimation of the system execution time. 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

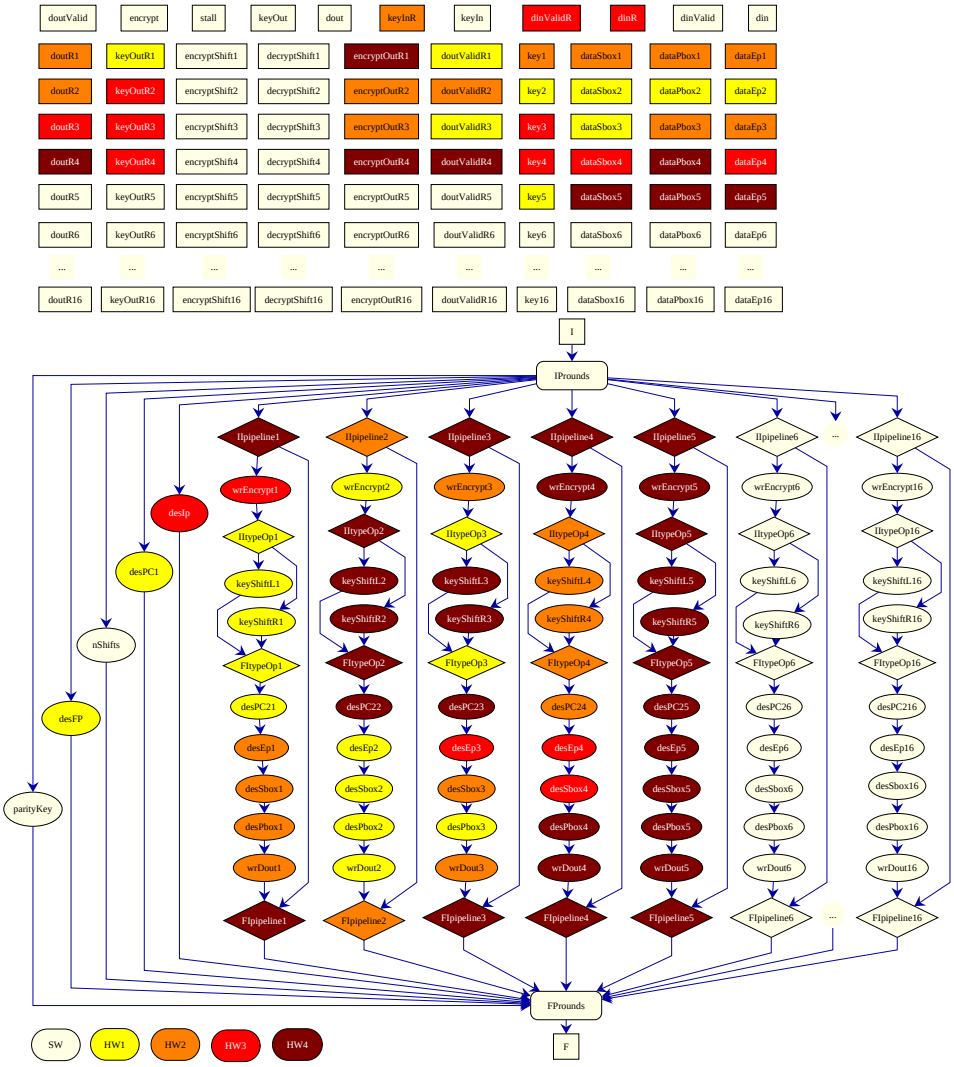

Figure 6: PSMfg model illustrating the best partition solution from the TS algorithm.

partition solution is high. However, in the majority of cases, the first searches of the partition process generate a solution with quality.

The support given by the partition methodology to the implementation of the systems was also evaluated. 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.

4 Conclusions

The cluster growth constructive algorithm follows a straight optimization heuristic, which proved to be able to generate solutions with quality, when guided by an adequate closeness function.

The results from the performed experiments with tabu search (TS) algorithm recommend that objects and moves tabu tenure must be 5 to 10% of the number of objects on the system description. To decrease the computation time while keeping the capacity to generate solutions with quality, the implemented TS algorithm only searches a partial neighbourhood, has a richer set of evolution strategies, applies a more efficient improvement and includes a richer set of diversification/intensification elements. To avoid the reduction of the capacity of converging to the optimum solution, the history of moves and moved objects must be reset by TS at the beginning of each search. A subset of tabu classifications was selected, which decreases the computation time, helps to avoid cycles and does not place excessive restrictions to the search. A neighbourhood with a simple structure also helps to decrease the computation time. The goal of the cost function applied by TS is to achieve the best partition solution with the available resources.

To generate accurate estimates, while keeping the computation time as low as possible, the implemented estimation methodology (i) uses detailed models for the hardware resources, (ii) runs in two abstraction levels and (iii) uses incremental updating.

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 respectively 82 to 98%, 92 to 99% and 89 to 96%. 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.

The estimated data for the interface resources and the communication time, simplifies the automatic synthesis of interfaces.

Some directions are being considered for future work: (i) evaluation of the methodology with more and differentiated case studies, namely more complex and control dominated systems must be tested; (ii) integration of the methodology on a broader one, which is used to develop concurrent systems that are implemented on a parallel, distributed and heterogeneous architecture; (iii) implementation of 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) optimization of 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] António Esteves. *A Partition Methodology for Digital Embedded Systems Codesign* (in portuguese). PhD thesis, Dep. Informática, Universidade do Minho, Braga, Portugal, July 2001.

[3] Asawaree Kalavade and Edward Lee. A Global Criticality/Local Phase Driven Algorithm for the Hardware/Software Partitioning Problem. In *Proceedings of the 3rd International Workshop on Hardware/Software Codesign*, pages 42–48. IEEE Computer Society Press, September 1994. Grenoble, France.

[4] J.M. Fernandes, R.J. Machado, and H.D. Santos. Modeling Industrial Embedded Systems with UML. In *Proceedings of the 8th ACM/IEEE/IFIP Int. Workshop on Hardware/Software Codesign (CODES'2000)*, pages 18–22. ACM Press, May 2000.

[5] P. Zave. The Operational versus the Conventional Approach to Software Development. *Communications of the ACM*, 27(2):104–118, February 1984.

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

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

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

[9] 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.

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

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

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

[13] Luís P. Santos and Alberto Proença. A Systematic Approach to Effective Scheduling in Distributed Systems. In *Proceedings of the 5th Int. Meeting on High Performance Computing for Computational Science (VECPAR'02), Porto, Portugal*, June 2002.

[14] Anton V. Chichkov and Carlos B. Almeida. An Hardware/Software Partitioning Algorithm for Custom Computing Machines. In *Field Programmable Logic and Applications - Proceedings of the 7th International Workshop FPL'97*, pages 274–283, September 1997.

[15] P. Middelhoek, G. Mekenkamp, E. Molenkamp, and Th. Krol. VHDL and CDFG Based Transformational Design: a Case Study. In *Proceedings of the ProRISC/IEEE Workshop on CSSP*, pages 203–212, March 1995.

[16] 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.

[17] 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.

[18] 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.

[19] Edna Barros and Augusto Sampaio. Towards Provably Correct Hardware/Software Partitioning using Occam. In *Proceedings of the 3rd International Workshop on Hardware/Software Codesign*, pages 210–217. IEEE Computer Society Press, September 1994. Grenoble, France.

[20] Rolf Ernst, Jörg Henkel, and Thomas Benner. Hardware-Software Cosynthesis for Microcontrollers. *IEEE Design & Test of Computers*, 10(4):64–75, December 1993.

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

[22] Petru Eles, Krkrysztof Kuchcinski, Zebo Peng, Alexa Doboli, and Paul Pop. Process Scheduling for Performance Estimation and Synthesis of Hardware/Software Systems. In *Proceedings of the 24th EUROMICRO Conference*, 1998.

[23] Peter Knudsen and Jan Madsen. PACE: A Dynamic Programming Algorithm for Hardware/Software Partitioning. In *Proceedings of the 4th International Workshop on Hardware/Software Codesign*, March 1996.

[24] L. Ferrandi, D. Sciuto, and M. Vincenzi. TOSCA User's Manual, Version 2.0, September 1997. `http://www.cefriel.it/eda/projects/seed/um/-mainmenuum.htm`.

[25] Petru Eles, Zebo Peng, Krkrysztof Kuchcinski, and Alexa Doboli. System Level Hardware/Software Partitioning Based on Simulated Annealing and Tabu Search. *Design Automation for Embedded Systems*, 2(1):5–32, 1997.

[26] Frank Vahid. Modifying Min-Cut for Hardware and Software Functional Partitioning. In *Proceedings of the 5th International Workshop on Hardware/Software Codesign*, pages 43–48, March 1997.

[27] Frank Vahid and Thuy Dm Le. Extending the Kernighan/Lin Heuristic for Hardware and Software Functional Partitioning. *Design Automation for Embedded Systems*, 2(2):237–261, 1997.

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

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

[30] Ricardo Machado, João Fernandes, António Esteves, and Henrique Santos. *Ch.11 An Evolutionary Approach to the Use of Petri Net based Models: from Parallel Controllers to HW/SW Co-Design*, pages 205–222. *Hardware Design and Petri Nets*, Ed. Alex Yakovlev, Luis Gomes e Luciano Lavagno. Kluwer Academic Publishers, Boston, USA, 2000.

[31] Daniel Gajski, Frank Vahid, Sanjiv Narayan, and Jie Gong. *Specification and Design of Embedded Systems*. Prentice-Hall, 1994.

[32] Daniel Gajski, G. Marchioro, and J. Zhu. *Essential Issues in Codesign*, pages 1–45. Kluwer Academic Publishers, 1997.

[33] António Esteves and Alberto Proença. A hardware/software partition methodology targeted to an FPGA/CPLD architecture. Submitted to *International Conference on Field-Programmable Logic and its Applications (FPL 2004), Antwerp, Belgium*, March 2004.