Partitioning and Exploration Strategies in the TOSCA Co-Design Flow

A. Balboni (a), W. Fornaciari (b, c), D. Sciuto (c)

(a) ITALTEL-SIT, Central Research Labs, CLTE, 20019 Castelletto di Settimo Milanese (MI), Italy

(b) CEFRIEL, via Emanueli 15, 20126 Milano, Italy,

(c) Politecnico di Milano, Dip. Elettronica e Informazione, P.zza L. Da Vinci 32, Milano, Italy

Abstract

The TOSCA environment for hardware/software co-design of control dominated systems implemented on a single chip includes a novel approach to the system exploration phase for the evaluation of alternative architectures. The paper presents the metrics and the partitioning algorithm defined for the identification of the best hardware and software bindings and modularization, given the design constraints and goals. The system exploration phase is implemented as an iterative process directed by the user, based on the formal internal design representation adopted in TOSCA. The application of the metrics is then shown on a simple example to illustrate the approach.

1. Introduction

In many embedded applications heterogeneous hardware/software architectures may provide a more effective design solution for some target cost and performance figures with respect to fully dedicated hardware implementations. Some of the most important aspects to be considered in the concurrent design of such systems is the early prediction of the final results, the possibility of generating different alternative solutions of the hardware/software partitioning process satisfying the user requirements, and the possibility of system designing by incrementing the functionalities of existing systems, i.e. the possibility of re-using existing modules (software or hardware).

The tools that implement such strategies should cooperate with existing design flows and standards to enable a realistic use within industrial design environments.

Aim of this paper is to introduce a novel methodology to manage the co-design process for a specific application field, i.e. control-dominated ASICs, such as those embedded into telecom digital switching subsystems. In particular, this paper will focus on the description of the implemented exploration strategies of alternative system architectures aiming at balancing hardware cost/performance and software flexibility.

Research in co-design has produced a number of different approaches ([1], [2], [3], [4]) which can be roughly partitioned in strategies starting from a fully software system implementation moving pieces of software toward the hardware domain and, *viceversa*, strategies aiming at obtaining the minimum cost by replacing pieces of hardware with software code. Our approach does not assume an initial solution, apart from the possibility of tagging some modules as software or hardware if they are already available in the design environment and no alternative solution is sought. It provides an environment enabling the exploration of different partitioning which can be either user-driven or automatically performed. All the decisions are evaluated and possibly taken according to the results of an iterative analysis performed through a set of metrics tailored to predict their effects onto the final implementation.

The paper is organized as follows. Next section gives an overview of the TOSCA framework and the target architecture adopted. Section 3 defines the internal system representation allowing the application of formally correct transformations of the system to identify and evaluate the different partitioning alternatives. The system exploration strategies and evaluation metrics are presented in Section 4. Finally an application of the partitioning algorithm is presented in Section 5.

2. The TOSCA design flow

The TOSCA environment represents a pragmatic approach to co-design, accepted as prototype in the R&D department of an Italian telecom company. In

0-8186-7243-9/96 $05.00

fact, the proposed framework is integrated within an industrial design flow and allows the user to approve or modify all decisions pertaining system exploration, with the possibility of backtracking along the design process and of reusing already designed submodules. The high level of integration with the existing commercial EDA tools has been obtained by directly interfacing with existing design entry environments (e.g. speed*CHART*), a VHDL representation of the hardware-bound parts, the direct synthesis of the software modules and the ability of achieving co-simulation of the entire mixed hw/sw system within the same VHDL-based environment. Moreover, the CAD environment allows the user to cover the whole design process, ranging from the system-level specification capture down to the synthesis, by considering low level effects of the software timing properties (down to assembly level), by providing the code for software processes as well as the necessary operating system support and, finally, by generating the interfaces among hw and sw modules.

Multiple formalisms are accepted as system specification, based on the consideration that in industrial design environment projects are seldom unconstrained but usually their specifications are derived from previous experiences, and therefore some components are a priori hw or sw bound, some are already synthesized, some are specified by graphical formalisms, others by textual specification. A unified internal representation level based on a process algebra computational model with an OccamII more pragmatic syntax has been adopted. Other modules, described through different formalisms, can be mapped or connected to the internal OccamII representation. The overall system representation is stored within an object oriented database tailored to support high-level architectural exploration, shared by all the tools composing the TOSCA environment.

The system exploration and partitioning strategy is based on the internal process algebra based representation, allowing the application of formal, well defined transformations. This process is managed by the Exploration Manager which drives manipulation of the initial system modularization to produce a new set of system partitions and their association (if still *floating*) either with software or dedicated hardware units. Our approach, apart from possible bindings forced by users, does not start from an *a priori* default solution (e.g. initially fully software bound). A set of strategies and basic algorithms can be iterated onto the system representation, until the design constraints are satisfied. The output of this process is a set of monolithic architectural units with a binding establishing either a hardware or a software implementation. Each architectural unit is then passed as input to the following co-synthesis stages.

The partitioning and synthesis processes have to consider different functional and non functional design requirements, that are tightly related to the target implementation architecture. Our class of applications requires a single chip implementation including an off-the-shelf microprocessor core with its memory (even if part of the memory can be external) and the dedicated logic implementing a set of coprocessors, i.e. the set of synthesized hardware-bound modules identified during system exploration. The term coprocessor includes also arithmetic/logic operations and possible private storage capability, while high-level synthesis tools typically separate controllers from data-paths. The master processor is programmable and the software can be either on-chip resident or read from an external memory; dedicated units operate as peripheral coprocessors. To satisfy the requirement of interfacing among hardware and software bound elements, a master-slave shared bus communication strategy has been adopted. All hardware to hardware communications are managed through dedicated lines (local buses). The RAM memory required for program/data storage shares with the coprocessors the main data bus, but can be accessed only by the master CPU. Communications among CPU and coprocessors are based on a memory mapped I/O scheme with one bus interface manager per coprocessor based on a common I/O buffered protocol manager.

The last stage defined within the TOSCA design flow covers the synthesis of all the elements: software-bound modules including the basic operating system support, hardware-bound parts and interfaces. The embedded system software is usually built around a lightweight operating system that provides process scheduling and interrupt handling facilities aiming at guaranteeing fairness and real-time behavior to all the *software actors*. Our solution is to consider the software description at the level of a virtual assembly instruction set (VIS) whose structure can be mapped onto different CPU cores with fully predictable translation rules and, consequently, reliable performance estimation [5].

This solution provides the possibility of achieving a fully VHDL based co-simulation of each proposed hw/sw architecture [5], to get feedback on the effectiveness of the implementation. In fact, a suitable VHDL model for VIS instruction-level execution, has been developed, so that the entire system can be co-simulated through a unified VHDL-based environment.

The model is parametric to allow the analysis of low-level timing/cost/performance, for different classes of microprocessors.

The hardware synthesis can be performed using three different strategies: direct mapping of hardware-bound FSM into a VHDL code compliant commercial RTL synthesis tools ([6]); transparent passing of imported VHDL modules descriptions to the logic synthesis tool; definition of behavioral VHDL descriptions of the hardware-bound modules to be input to a commercial high-level synthesis tool.

3. Design representation

The internal model onto which the system exploration model is defined is based on a customized process algebra/OccamII representation. The system representation can be obtained by importing modules descriptions obtained through commercial design entry tools (such as speed*CHART*) as well as by using the built-in OccamII editor (see fig.1).

A key issue of the OccamII formalism is the possibility of easily representing both parallel and sequential execution of processes at any abstraction level. For example, by considering a pair of processes p2 and p1, the termination of p2-after-p1 execution can be captured by the construct SEQ, while and the impossibility of statically predicting the termination order between p1 and p2 is expressed via the PAR construct. It is important to underline that no restriction on the process granularity is required, so that processes executed in parallel can be constituted by assignments, input or output statements as well as parallel or sequential composition of simpler processes. Details on the Occam paradigm can be found in [7], [8].

The main advantages provided by the chosen process algebra/OccamII for co-design purposes, with respect to the other formalisms (e.g. the *statechart* family, flow-chart diagrams, CDFG, VHDL, SDL, ...) are the following:

- the computational model is based upon a small set of primitives (assignment, input and output) and composition operators (sequential, parallel, alternative) allowing the modeling of arbitrary complex behaviors;
- the simple and regular syntax makes easier the parsing, the redisplay of transformed specifications and the internal representation;
- a formal and well assessed theory of transformations is available;
- multiple paradigms (event-driven, dataflow, rendez-vous, FSM,...) are supported through the same syntax and semantics;
- a powerful concurrency model suitable for system-level specification is defined: channels, user-defined communication protocols, regular parallelism (array of processes and channels), timers and timeouts, process priorities.

An important advantage of using channels to represent communication is the possibility of applying the same compact model of data exchange between heterogeneous processes (and consequently interfaces). The channels behave as variables with an attached *unidirectional pipe* management, leaving unmodified the original value (the channel receives only a copy). Since a channel can be shared by two processes only, broadcast communication can be modeled via multiple channels. Synchronization among processes is realized through channels, because they implement a rendez-vous mechanism. Shared variables among parallel processes are forbidden, they can only transfer values via channels.

Time related issues are supported through the use of *timer* objects, acting as read-only channels providing an integer value corresponding to the current time.

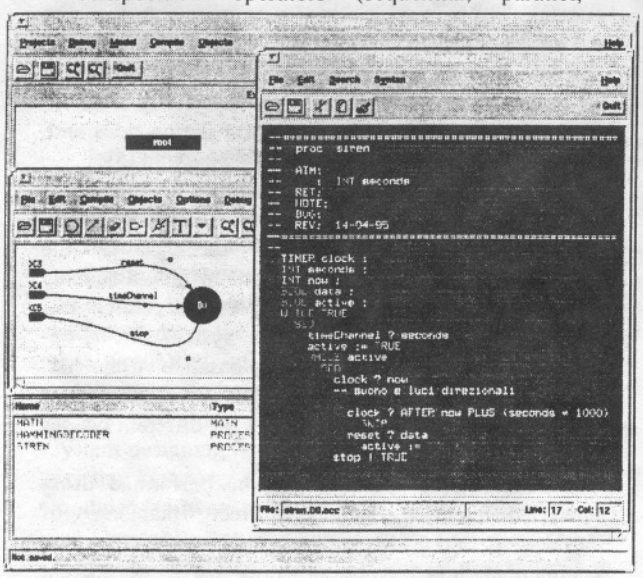
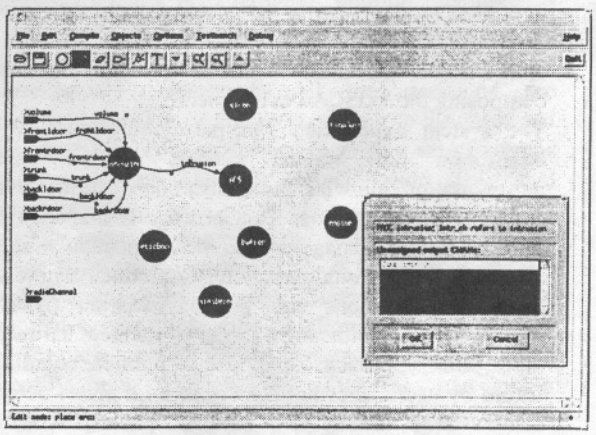


Figure 1: The visual front-end of the TOSCA environment.



Delays can be introduced by using deferred inputs through the AFTER clause. Typical real-time schemes can be modeled by combining the statements for managing time with the construct allowing alternative composition of processes.

The Occam model of the system description is stored within the TOSCA object oriented design database, storing classes of objects for modeling the system functionality and supporting the specific co-design activities such as simulation and design entry. All the classes inherit from a top-level class the following methods:

- general purpose methods implementing functionalities for database management;
- methods realizing a user-level visual back-annotation interface of information concerning the objects;
- methods activating specific actions operating on the associated object.

4. System exploration and partitioning

After the acquisition of the system model has been completed, architectural tradeoffs may be carried out by iterated manipulation of the internal model stored within the TOSCA database. This stage manages two classes of objects: *processes* and *architectural units*. The processes can be subdivided in two classes:

- processes to be mapped onto predefined hardware or software components;
- processes whose specification is not biased by the implementation and therefore different design alternatives can be explored at system level (e.g. the ones captured via the graphic OccamII built-in editor of TOSCA).

The partitioning process can be viewed as an incremental activity of modification of the initial specification through the application of transformations. The TOSCA project has considered this process through:

- The definition and implementation of formal transformation rules working onto the internal model stored within the design database.
- The definition and implementation of a set of metrics to drive the partitioning process.
- The definition of a partitioning algorithm and its implementation within the Exploration Manager framework to obtain a tool allowing both direct intervention of the user and automatic selection of the strategies by following the built-in evaluation criteria.

Since Occam has been derived from CSP [8], a number of formal properties are available as a background to define a set of transformations preserving the semantics of the original specification.

Some preliminary studies on transformations tailored for state-based synchronous specifications have been experimented in [6], but the current version of TOSCA includes:

- Parallelization of sequential processes and viceversa, that can be useful to improve performance or reuse of basic building blocks (hw o sw) of processes belonging to the same architectural unit.
- Replacement of channels with variables and viceversa; this action is usually performed when two processes will become part of the same sw-bound architectural unit. For instance, processes belonging to the same coprocessor share the local variables while if they are located on different coprocessors communication channels are required.
- Merge of code sections.
- Splitting of code sections.

The partitioning process can be executed step by step, thus allowing direct control of the user through the exploration manager interface. This allows to take advantage of user expertise whenever possible, but it can be automatically performed by applying the default partitioning flow. Here, the default partitioning flow is introduced.

The top-level algorithm of the default flow uses a *greedy* strategy where the initial model is first of all expanded in a top-level process containing a PAR assert, by properly applying the formal transformation laws. The system allows modification of the granularity of the decomposition according to the user selection. After this preliminary step an initial graph G containing all the not yet allocated processes (leaf nodes) is built. A set of actions is applied onto such a graph. These actions can be partitioned in two phases:

1. Pre-allocation: this phase consists of marking the processes nodes according to the most suitable implementation technology (hw or sw). The user can accept the proposed solution or manually modify the decisions taken.
2. During a second phase, according to some closeness criteria defined by the user and based on the metrics available, new process *leaves* are built by pairwise collapsing nodes to build a unified PAR statement; this activity of collapsing nodes is iteratively performed until all the architectural units are identified and their binding is defined.

The initial pre-allocation strongly affects both the convergence speed of the algorithm and the result quality. The collapsing of processes in the second phase, will occur whenever the measure of the closeness criteria, during the selection of the closest processes, stays under a certain *a priori* defined threshold. This solution is particularly flexible since it is possible to consider a set of criteria, with different priorities that can be changed dynamically, according to the current nodes granularity or user choice.

To support these activities, a set of metrics has been defined, according to the co-design target. Our approach focuses on providing a modular design flow

sufficiently open to be easily extended to cover any new strategy defined. For this purpose the mechanisms implementing the system transformations have been kept disjoint from their application policy. A specific set of metrics for early prediction of cost/performance, as well as to evaluate the system synthesis results, has been developed at first. The activity of identifying some closeness properties among parts of the system specification, can be performed by means of a static analysis based on metrics for early prediction of cost/performance. It aims at producing an initial nearly-optimum allocation and binding to be iteratively improved by the user until design goals are satisfied. Afterwards, to update the predicted data, a final stage of actual synthesis is performed. This is the most time consuming task and it is strongly sensitive to the quality of the pre-allocation performed during the former phase; it returns information that will be back-annotated as replacement of the predicted data.

The metrics consider the system analysis from a threefold point of view:

- *Statically*, by analyzing composition and structure of the description contained within the database.
- *Dynamically*, but still independent of the implementation, through an high-level execution and profiling of the specification to extract information such as communication bottlenecks and statistics on operators applications to better tune the decision on the initial partitioning and binding.
- *After-synthesis*, requiring at least a complete synthesis cycle. This returns information that will be back-annotated as a replacement of predicted data and will also contribute to the final design evaluation.

The application of metrics is based on the association of each Occam process with a module, including the exported interface and the process body that is hidden to the other processes. Under a structural point of view, each Occam module composing the system description can be considered as a hierarchy of modules, being each component a module itself.

The most relevant factors considered on such modules to evaluate the quality of the resulting design, are the following.

- *Communication*: costs related to the number of lines and bandwidth for hw-hw and sw-hw communication.
- *Interfacing*: according to the adopted communication/synchronization technique among different modules (e.g. between hw and sw, one interface unit per coprocessor vs a common bus manager/arbiter queuing messages), costs can be affected by number and granularity of modules. Alternative bus protocol templates can be evaluated.
- *Area*: this is the most important aspect because of the single chip implementation. Overall optimization takes into account possible user-defined binding along the evaluation and comparison among different alternatives.
- *Resources exploitation*: on the software side, since the microprocessor is anyhow present, it is important to increase as much as possible the CPU utilization while fulfilling the timing requirements of the programmed modules and the effectiveness of memory. Another relevant issue is related to the power consumption that is improved by a modularization able to identify the minimum set of architectural units with the lowest amount of idle time.

Currently, the TOSCA environment implements a set of methods to evaluate the following static metrics, based on the following static properties of the Object Oriented representation of the Occam specification:

- Declarations count.
- Data dependencies among data declarations.
- Module use.
- Structure of modules composing the system graph.

The relevant parameters considered to determine the declaration count are:

- Local(module): representing the number of local data declarations.
- Global(module): representing the number of data visible within a module.
- Scope(module): representing the visibility area of data declaration overlapped to the module definition.

The properties representing the relations among modules are captured by considering the number of other modules both used (*imported*) and using the considered (*exported*) one. Other important characteristics are obtained through the analysis of the graph representing the hierarchy of the modules composing the system. In particular maximum and average depth and the number of possibly complete paths (from the root to a leaf) are taken into account. These parameters can be considered as a bias to estimate the system complexity.

Another set of metrics has been introduced to report the presence of possible inconsistencies when modules are collapsed. They describe the level of internal cohesion for each module and the coupling between modules. Two types of interactions have been evaluated: *declaration and declaration* which analyzes if a declaration interacts with another by considering if a modification of the first implies also a modification in the second one; *declaration and subroutine* which considers that a declaration interacts with a subroutine if it has a declarattion and declaration interaction with at least one of the subroutine declarations.

A significant goal of the entire partitioning process, is to achieve an high level of internal cohesion together with a loose coupling with other modules. This is particularly important e.g. to make easier the moving of communicating modules within different implementation domains while saving bus bandwidth,

since it is the only mean to obtain hw-sw data exchange. Moreover, it is also possible to avoid unnecessary replication of data variables (or registers for the hw domain) to implement the corresponding data handler.

In addition to the static properties, a profiling through simulation of the internal specification can be carried out. This allows to better understand the real bottleneck and critical parts of the system description. Since the purpose of this analysis is to provide statistics and predictions of the execution times, a parametrizable model has been defined to cover both the hardware and software domain. In fact, a process composed of basic actions cannot be represented by simply adding the execution times of each component. It is necessary to consider also the effect of process synchronization and communication that are non explicitly managed by Occam descriptions. For instance, if software execution is considered it is necessary to model the overhead due to context switching among threads and the presence of subroutine calls.

This modeling of the time properties, allows the identification of the following process properties:

- Latency of a process execution.
- Execution frequency of a process.
- CPU usage (summation of the latency-frequency products, for all the software bound processes).
- Channel utilization (bandwidth, idle time).
- Computational load, representing the total number of operations executed by process p during simulation.

All the data obtained through metrics evaluation are gathered within matrices that will be used to evaluate the closeness among the processes composing the system graph, and possibly to trigger the application of system-level transformations, according to a hierarchical multi-stages clustering algorithm. In general, the cost function used to evaluate the quality of candidate solutions has the form of a weighted sum:

$$\text{cost} = \sum_{m \in M} w_m \, m = W \times M$$

where M is the set of the considered metrics and W the matrix expressing the relevance of the metric according to the designer's goal. In case of direct intervention by designer, the results of metrics evaluations can be back-annotated and considered separately. The evaluation of metrics computed onto the design description stored within the database is basically tailored to support the system level exploration phase, i.e. as a support in the process of collapsing processes to identify a set of architectural units with the proper hw or sw binding and to modify the process granularity.

The algorithm operates by considering the set G of processes constituting the system graph and the closeness matrix $\mathbf{V}(c,G)$ according to the criterion c considered; in other words, each $\mathbf{V}[i,j]$ entry represents the measure of the distance between nodes i and j based on the evaluation of the c criterium. The decision on the processes to be collapsed is carried out by the functions *coupleWithMaxCloseness(V)* which selects the closeness pair of nodes of the graph, and *maxCloseness(V)* which returns the corresponding distance. Given a certain criterion, the opportunity to collapse nodes is driven by the crossing of a user defined threshold Vt. The decision on whether or not the nodes have to be collapsed is related with the threshold Ct defined onto the cost function summarizing the adherence between design results and initial requirements. This method allows to establish an ordering in terms of importance among the considered criteria, which can be modified dynamically. A pseudo-code representation of the algorithm is here reported:

```
while costFunction > Ct {
  select c in criteria
  while maxCloseness(V(c,G))≥ Vt(c) {
    (i,j) := coupleWithMaxCloseness(V(c,G))
    G1 := G - {i,j}
    G2 := G1 + { collapse(i,j)}
    if not G2.violatesConstraints() {
      G := G2
    }
    modify (the system)
  }
}
```

The procedure *modify* allows the introduction of some modifications into the system model such as a new set of bindings, the serialization of operations within processes obtained through previous collapsing and so on. *G2violatesConstraints* evaluates if the considered system instance is acceptable according to the defined constraints; to reduce the possibility to be cached in a local optimum, a non deterministic function can be introduced. The complexity of the implemented algorithm is $O(c \times n \times \log_2 n)$ where c is the number of criteria, n is the number of processes of the system.

The derivation of the closeness matrix starting from a given metric can be performed by following two different approaches according to the type of considered metric. Metrics can express properties both intrinsic the module (e.g. the Local(m) metric) and related with its connection to the system (e.g. the Exported(m) metric) or concern the interaction between pairs of modules (e.g. the Contemporaneous Presence ($m1$, $m2$)).

In the first case a vector $\mathbf{M}(c,G)$ is built such that each element $\mathbf{M}[g]$ represents the evaluation of metric c for the process $g \in G$. For each pair of processes $\{i,j\}$, the values of matrix $\mathbf{V}[i,j]$ are given by:

$$\mathbf{V}[i,j] = \frac{\mathbf{M}[i]}{\mathbf{M}[j]} \text{ if } \mathbf{M}[j] \geq \mathbf{M}[i],$$

$$\mathbf{V}[i,j] = \frac{\mathbf{M}[j]}{\mathbf{M}[i]} \text{ if } \mathbf{M}[i] > \mathbf{M}[j].$$

For the latter case the distance matrix is obtained by normalizing the entries of $\mathbf{M}(c,G)$ so that each of its elements $\mathbf{M}[i,j]$ represents the evaluation of metric c with respect to the processes i and j. As a consequence:

$\mathbf{V}[i,j] = 1$ if $\{i,j\}$=coupleWithMaxCloseness($\mathbf{M}(c,G)$)

$$\mathbf{V}[i,j] = \frac{\mathbf{M}[i,j]}{\mathbf{M}[k,m]}$$

if $\{k,m\}$=coupleWithMaxCloseness($\mathbf{M}(c,G)$).

If a criterion is associated with more than a single metric, the distance matrix is obtained by building a matrix whose elements are the weighted sum of the corresponding entries belonging to the matrices concerning the considered metrics (one per metric), and finally by normalizing the elements to the highest value obtained. The entries of the matrix all range from 0 to 1, so that the same range of values can be considered for the threshold Vt.

For example, by considering for simplicity only the cohesion metric, i.e. the metric which evaluates the degree of interrelation between two modules, it is possible to define a threshold that, if exceeded, enables the collapsing of two processes *p1* and *p2* within a single module *m*. This choice, according to the considered metric, is advantageous if the cohesion of *m* is greater than the cohesion of the set constituted by modules *p1* and *p2*. In other terms, the decision depends on the following relation:

$$\frac{CI(m1)+CI(m2)}{M(m1)+M(m2)} > \frac{CI(m1)+CI(m2)+CI_{12}}{M(m)}$$

where *CI(mi)* represents the number of cohesive interactions of module *mi*, *M(mi)* is the maximum number of possible cohesive interactions of module *mi* and CI_{12} expresses the number of cohesive interactions belonging to *mi* and *m2* when they are collapsed. The values of CI_{12} obtained by solving the above relation, can be considered as the threshold for the cohesion metric.

It has to be pointed out the high flexibility of the proposed approach that can be extended with minimal re-design effort to cover new design goals or characteristics, simply

- by adding a proper method operating onto the design DB to compute the corresponding metric;
- by defining its relative importance with respect to the other existing metrics in case of user-driven design to produce a consistent set of analysis data evaluating the system quality, as well as if the automatic system partitioning is enabled to modify the selection strategy.

5. Example of the partitioning process

Let us consider a small example to show the default partitioning flow, using only static properties due to space reasons. The system implements a convolver whose functionality is expressed by the following equation: $y_i = \sum_{j=1}^{n} x_{i-1} \times w_j \times \alpha_i$

where $1 \le i \le 2n-1$, $w_{j+1} = b \times x_1 \times w_j$ and α_i is given by $\alpha_i = c_i + d_i$ with $c_i = d_i$ *if* $x_i \ge 0$, $c_i = x_i/2$ if $x_i < 0$ and $d_i = x_{i+1}$ if $x_{i+1} \ge 0$, $d_i = (x_{i+1})/2$ if $x_{i+1} < 0$.

The corresponding Occam representation, stored within the TOSCA design database, is composed of three processes whose OccamII hierarchy and interactions are depicted in figure 2:

p0 models the environment and produces the values for *x* and *w*;

p1 implements the convolver;

p models the parallel composition of p0 and p1.

The communication between p0 and p1 is performed via channels.

The system analysis starts by taking into account for simplicity only the metric measuring the number of paths of the overall system (Paths) to define the cost-function weighted sum (i.e. the coefficient multiplying the other metrics are zeros) and to accept solutions producing *cost* < 35.

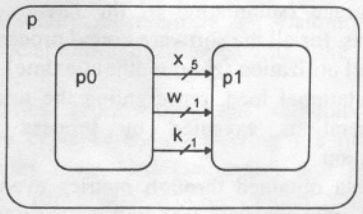


Figure 2: Schematic diagram of the example.

Therefore, the cost function is: *cost* = *k* * *Paths* (with k=1). In addition we introduced some limitations on the metrics measuring the number of variables local to each process (that are related to the size) and to the process latency. The corresponding constraint for each process *p* is thus: *(Latency (p) < 12) AND (Local (p) < 20).*

A static analysis of the system model considered as a whole has been carried out, the result of the static metrics evaluation shows that the system does not satisfy the above conditions (*Paths* evaluates 43 that is greater than 35) so that it needs to be restructured to fulfill the user's goal. The first step has been to consider the system at a finer granularity by decomposing p1 into a set of parallel processes, whose structure is shown in figure 3.

To identify a better system modularization, a set of metrics has to be evaluated. Here, only five static metrics have been computed. The data obtained after processing the system description are reported in fig.4. Based on these results, for each metric the corresponding closeness matrix between processes can be derived. For instance, the matrix corresponding to

the *local(p)* metric is shown in fig.5.

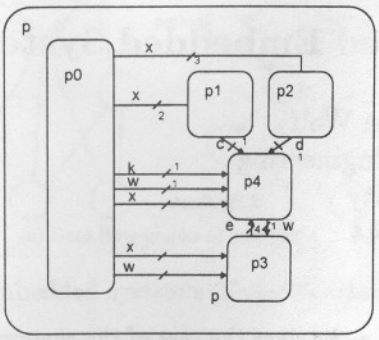


Figure 3: The system after the first decomposition.

Property/proc.	*p1*	*p2*	*p3*	*p4*
Local	3	4	10	17
Global	3	4	10	17
Depth	3	3	3	4
Paths	4	5	18	16
Cohesive Inter.	1	1	5	5

Figure 4: Results of the analysis of five properties of the system description.

Similarly, according to the data contained in figure 4, it is possible to compute the values for each one of the considered metrics. All these values are then gathered within a matrix which is evaluated to compute the closeness among the processes composing the system graph.

By considering as global closeness criterion only the *local* metric, processes p1 and p2 in figure 3 are the closest and their collapsing does not violate the constraints.

LOCAL	*p1*	*p2*	*p3*	*p4*
p1	-	1	.4	.24
p2	-	-	.53	.34
p3	-	-	-	.79
p4	-	-	-	-

Figure 5: An example of computation of the local *metric. The values, representing the number of declarations local to a module, have been normalized.*

Since p1 and p2 show a low level of cohesive interaction, they can be composed to produce a unique software-bound process. To allow their execution on a single microprocessor, a sequential composition of the original pair of processes has to be applied. The transformation process proceeds in a similar way and processes p3 and p4 are collapsed. According to a profiling obtained via a dynamic analysis, the corresponding architectural unit is hardware bound since it accounts to 73% of the overall computational load. This modularization corresponds to an acceptable and final redesign of the system, since a new computation of the metrics produces a set of values satisfying user's goal and constraints:

cost = 28 < 35 latency =9 < 12 AND local =17 < 20

The partitioning and binding process is thus completed and the system is ready to be processed by the TOSCA co-synthesis tools to produce VIS and VHDL code for the p12 and p34 architectural units, respectively. Eventually, the channel used for communication among processes belonging to the same partition are replaced with global-scope shared variables and a serialization of the operations of all the sw-bound processes is automatically performed.

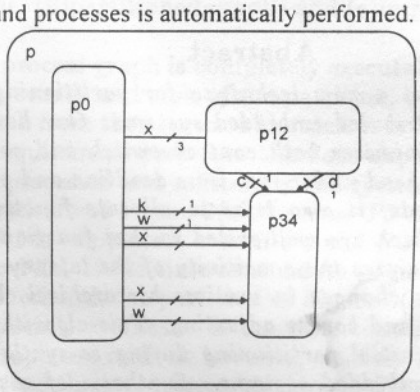


Figure 6: The final system modularization after collapsing of p3 and p4.

References

1. Hardware/Software Co-Design, NATO Advanced Study Institute, G. De Micheli, M.G. Sami eds., to appear by Kluwer, 1996.
2. Benner T., Ernst R., Henkel J., "Hardware-Software Cosynthesis for Microcontrollers", IEEE Design&Test, Vol.10, No.4, December 1993.
3. Barros E., Rosenstiel W., Xiong X., "Hardware/Software Partitioning with UNITY", Proc. 2nd Workshop on HW/SW Co-Design, Cambridge, MA, 1993.
4. Ismail T., O'Brien K., Jerraya A., "Interactive System-level Partitioning with PARTIF", Proc. EDAC'94, Paris, France, February, 1994.
5. Antoniazzi S., Balboni A., Fornaciari W., Sciuto D., "The Role of VHDL within the TOSCA Co-design Framework", Proc. of Euro-VHDL'94, September 1994.
6. Antoniazzi S., Balboni A., Fornaciari W., Sciuto D., "HW/SW Co-design for Embedded Telecom Systems", Proc. ICCD'94, pp.278-291, 1994.
7. Jifeng H., Page I., Bowen J., "Towards a Provably Correct Hardware Implementation of Occam", Technical Report, Oxford University Computing Laboratory, 1994.
8. Hoare C. A. R., "Communicating Sequential Processes", Communications of the ACM, Vol. 18, No. 8, 66-77, August 1978.