Microprocessors and Microsystems 38 (2014) 9–21

Contents lists available at ScienceDirect

Microprocessors and Microsystems

journal homepage: www.elsevier.com/locate/micpro

A two-phase design space exploration strategy for system-level real-time application mapping onto MPSoC

Z.J. Jia [a,*], A. Núñez [a], T. Bautista [a], A.D. Pimentel [b]

[a] Institute for Applied Microelectronics, University of Las Palmas de Gran Canaria, Campus Universitario de Tafira, 35017 Las Palmas de Gran Canaria, Spain
[b] Computer Systems Architecture Group, Informatics Institute, University of Amsterdam, Science Park 904, 1098 XH Amsterdam, The Netherlands

ARTICLE INFO

Article history:
Available online 9 November 2013

Keywords:
Computer-aided design
Performance analysis
MP-SoC design
Experimentation
System-level design space exploration
Mapping strategy

ABSTRACT

In this paper, we present a two-phase design space exploration (DSE) approach to address the problem of real-time application mapping on a flexible MPSoC platform. Our approach is composed of two independent phases – analytical estimation/pruning and system simulation – communicating via a well-defined interface. The strength of the resulting strategy is twofold. On one hand, it is capable of combining the benefits of analytical models and simulation tools (i.e., *speed* and *accuracy*). And on the other hand, separating pruning and evaluation phases facilitates the integration of different or additional pruning techniques as well as other existing simulation tools. Finally, we also present several proof-of-concept DSE experiments to illustrate distinct aspects and capabilities of our framework. These experimental results reveal that our approach, compared to other approaches based only on analytical estimation models or simulations guided by e.g. genetic algorithms, not only can explore a large design space and reach a valid solution in a time-efficient way, but also can provide solutions optimizing resource usage efficiency, system traffic and processor load balancing.

1. Introduction

The increasing levels of integration are leading to more complex embedded systems on chip, which can contain multiple and possibly different processing elements, storage elements and network elements [1]. In this context, these multiprocessor-systems-on-chip (MPSoC) platforms are emerging as an interesting solution in the development of modern SoCs, since they can provide a flexible and re-usable architecture to support different product versions (or family of products), and an architecture that can be easily modified in response to market needs, users requirements and product updates during the product life cycles.

In order to cope with the challenge of designing such complex MPSoCs under an increasing time-to-market pressure and enormous market competitiveness, Design Space Exploration (DSE) is becoming a key ingredient for system designers. DSE consists of exploring a large number of different design decisions (or options) to find a set of optimal system designs that matches the user specifications and design constraints (such as power/energy, cost/area, and throughput). That is, DSE for modern embedded system design often has to cope with a multi-objective optimization problem.

Different kinds of DSE can be carried out during the whole system design process (from the initial specifications to the final design implementation). An example of DSE classification is shown in Fig. 1, although other classifications can be also found in the literature [2,3]. The DSE classification illustrated in Fig. 1 is based on the range of design options explored at different abstraction levels, where these DSE processes are orthogonal to each other. For example, a DSE in the application domain can explore different possibilities of partitioning an application at algorithmic level, while the exploration of the mapping design space may comprise design decisions such as HW/SW partitioning and task assignment on processing resources. On the other hand, architectural platform DSE usually concentrates on overall system design (e.g., decisions about the number and type of components, component allocation in a platform, etc.) instead of the individual components, while micro-architecture DSE is focused precisely on exploring this in-component configuration space, i.e., refining component parameters such as cache size, bus arbiter policy, and functional units within a processor architecture. In this paper, we focus on both mapping and architectural platform DSE at system level.

Independent of which of the aforementioned design spaces is explored, any DSE process can be decomposed into two interdependent components [4,5]: the strategy for exploring (i.e., searching) the design space and the mechanism for evaluating a single design point in that space. In fact, these DSE components have received significant research attention during the last decades [3,5–17], since they both heavily influence the efficiency and accuracy of the DSE process.

* Corresponding author. Tel.: +34 928 451230; fax: +34 928 451083.
E-mail address: cjia@iuma.ulpgc.es (Z.J. Jia).

0141-9331/$ - see front matter © 2013 Elsevier B.V. All rights reserved.
http://dx.doi.org/10.1016/j.micpro.2013.10.005

The ultimate goal of exploration strategies is to visit different design points in a design space in order to find a set of solutions that best optimizes the multiple and often contradictory design constraints. These exploration strategies can be divided into two categories: exact and approximated approaches. An exact approach (e.g., exhaustive search) is only possible when all the points in a design space have been characterized in terms of the target metrics or design objectives. However, the main problem is that the design space often is prohibitively large for an exhaustive search. In fact, the higher the abstraction level of the DSE, the more design options can be explored, and the more design alternatives can be chosen to satisfy a given set of design constraints (Fig. 1).

When an exhaustive search is infeasible, approximated exploring approaches must be adopted. Basically, an approximated approach (e.g., random search, techniques based on evolutionary algorithms and/or heuristics, etc.) proposes to prune the design space by exploring only a limited number of design points. Although such design space pruning indeed improves DSE efficiency by reducing the overall analysis time [6–13,18,19], it is important to note that this approach does not guarantee that optimal solutions are found. Instead, it will identify a set of near-to optimal (or sub-optimal) solutions.

The efficiency of the DSE process can also be improved from the perspective of the method used to evaluate design points. As depicted in Fig. 1, the evaluation method applied in DSE typically depends on the particular abstraction level and design objectives. For example, at higher levels of abstraction, estimation techniques (ranging from analytical models to system-level cycle-approximate simulation) allow designers to rapidly obtain estimates of the final characteristics of a design point. At lower levels of abstraction, evaluation tools such as Instruction Set Simulators and RTL simulators are capable of carrying out slow(er) but cycle-accurate analysis. As this work addresses system-level DSE, the rest of the paper will focus on analytical and cycle-approximate simulation techniques.

The low computational demands of an analytical model enable it to analyze a large number of design alternatives in a reasonable amount of time. However, it also suffers from two drawbacks. First, most analytical models only focus on the importance of a subset of design variables, while they assume fixed values for other design options and/or do not model some design parameters that form the axes of the design space [3,7,9–14,18,20]. As a consequence, the resulting analytical model is often inflexible and hard to reuse for different case studies. Second, analytical models often fail to consider *non-deterministic* system behavior, such as unpredictable arbitration delays of communication architectures due to simultaneous access requests by multiple competing processors. As a result, such *non-linear* system behavior makes an accurate performance estimation difficult without the use of simulation.

System-level simulation, on the other hand, allows for accurately evaluating the dynamic system behavior and provides detailed system metrics [15–17]. However, it is impractical to use these simulators to explore large design spaces due to (i) the high set-up effort for creating different architecture and mapping models, and (ii) the prohibitively high total run-time. As a consequence, the simulation tools are often coupled with some kind of pruning approach in practice [6,9–14,18,21]. However, in spite of the efficiency improvements achieved in those approaches, it should be noticed that the total run-time of such DSE experiments is usually still in the order of several hours.

Based on the former analysis, it is clear that efficient DSE for application mapping onto MPSoC platforms is a non-trivial problem. First, in order to find a set of optimal mappings under multiple design constraints, a vast number of design decisions should be explored. Not surprisingly, this problem is *NP-hard* [22]. Second, both analytical models and system-level simulators are not suitable

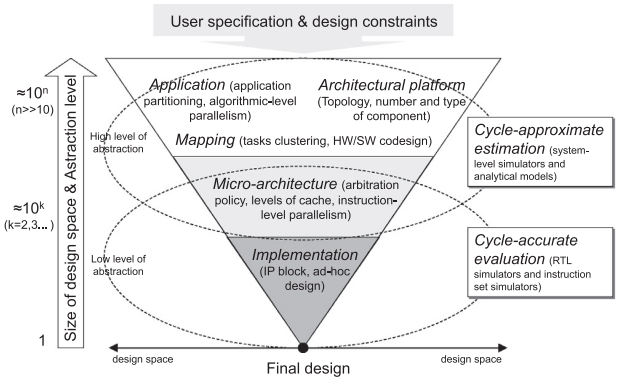


Fig. 1. An example of classification for DSE approaches.

enough to explore such a large design space of mappings, since the former is not sufficiently accurate and the latter are too time-consuming.

This work presents a DSE strategy that addresses the above problem. Our final goal is to find the optimal mapping(s) of a real-time application on an MPSoC platform while considering multiple design objectives. To this end, our approach is composed of two phases – analytical estimation/pruning and system simulation –, such that the resulting strategy is capable of combining the benefits of both analytical models and system-level simulation tools (i.e., *speed* and *accuracy*). In the first phase, we use a set of analytical methods considering both deterministic and dynamic system behaviors to rapidly explore design points, as well as to eliminate those design points that cannot satisfy the user requirements. The output of this pruning process is a reduced number of mapping alternatives, which are evaluated accurately in the second phase by a system-level simulation tool in order to find the (sub-)optimal mapping solution(s). The strength of our approach lies in its flexibility. Separating the pruning and evaluation phases facilitates the integration of different or additional pruning techniques as well as other existing simulation tools, if they satisfy the simple interface requirement of our framework.

The remainder of the paper is organized as follows. In the next section, related work is surveyed. Section 3 provides some concept definitions used throughout this paper, as well as presents an overview of our approach. Subsequently, Section 4 describes various implementation issues of our approach. In Section 5 we present a range of experimental results from a case-study with a Visual Tracking Application. And finally, Section 6 concludes the paper and discusses future work.

2. Related work

There exist numerous efforts on DSE for MPSoC design, of which only a representative subset will be discussed in this section. Particularly, we will put emphasis on approaches addressing the problem of application mapping on MPSoCs, as well as on strategies for achieving an efficient DSE.

Jia et al. [18] propose a two-phase approach based on heuristics to explore distinct mappings of a real-time application on a homogenous multiprocessor architecture. Kim et al. [23] also proposed a framework based entirely on analytical methods for DSE. In this case, they focused on the exploration of a design space associated to the HW/SW partitioning and scheduling policy for each processing element, in order to ensure that the application timing requirements are met. In the framework presented by Madsen et al. [13], different mapping alternatives are evaluated (by means of an

analytical model) for a flexible platform during the exploration process. All these works are based on simplified communication architecture models to estimate the system traffic cost, which may not be realistic enough for modelling modern embedded systems, as they may lead to non-optimal solutions.

In order to overcome this lack of realism of the modelling of communication, analytical approaches considering the dynamic behavior and resource sharing have been proposed in [20,24]. Basically, they propose an analytical model for each kind of architectural component that can later be composed to capture and analyze the complete system. However, the authors of these efforts also state that, in spite that their models can achieve a good level of accuracy (with a reasonable estimation error), it is not easy to combine such models for the evaluation of large systems such as those required in many MPSoC applications.

A system-level simulation tool represents a natural alternative for a more accurate performance evaluation, and/or the analysis of more complex design problems in larger embedded systems. In [6], Sesame [16] was coupled with genetic algorithms (provided by the PISA framework [3,25]) to address the mapping of multiple applications (with different timing requirements) executing concurrently on the same MPSoC. In [19], CASSE [15] was used in DSE experiments aiming at finding the optimal mapping of a target application on an MPSoC that works with multiple clock domains. Lahiri et al. [7] combined PTOLEMY [17] with an iterative algorithm to determine the well-optimized configuration of arbitration schemes for the different network elements on the architecture, such that the system performance is maximized. However, in spite of achieving the desired solutions with a relatively low number of simulations, a high total run-time – typically in order of hours – of each DSE experiment still is a common denominator in these works.

A few approaches combining analytical estimation models and simulations in a single framework have been proposed. Most of them are focused on the problem of micro-architecture DSE [9–12], and their ultimate goal is to minimize the number of simulations executed during the DSE process. To this end, several novel techniques have been proposed (e.g., estimation based on a hierarchical fuzzy system [9], response surface modelling [10], statistical trace-driven simulations [11], and heuristics [12]), which are often coupled to evolutionary algorithms to prune the large design space and provide an acceptable approximation of the Pareto-optimal front, i.e., a set of sub-optimal solutions. These works are orthogonal to our approach, as we explore the mapping and architectural platform design spaces at a higher level of abstraction, and the above approaches can be applied to further refine the parameter configurations of components used in the architectures found in our DSE.

Finally, a hierarchical and three-phase DSE methodology was explained in [8], which has a similar objective as the approach presented in this paper. However, there are some key differences with respect to our work. While our approach deploys an analytical estimation model to carry out the pruning phase, the approach presented in [8] draws a clear boundary between the pruning process and system performance estimation. That is, their approach first uses a symbolic constraint satisfaction method to identify candidate design points meeting the user specified constraints, and subsequently, the system wide performance of the design points obtained in that pruning process is individually evaluated using simulations. On the other hand, unlike our work, the impact of the resource placement in the architecture, as well as the contention effects on the shared communication architecture are not explicitly explored in [8].

Lee et al. [14] also presented a framework that determines the optimal mapping of a given real-time application for MPSoCs. This two-phase approach first selects an optimal set of processing elements for the mapping of the target application. Then, by means of a static estimation method based on a queuing model, they explore and prune the design space of communication architectures. And finally, their framework provides a set of interfaces that facilitate the integration of different simulation tools, which can be used to accurately evaluate each solution in the reduced design space. Unlike our approach, they address the DSE as a high-level synthesis problem. Moreover, their queuing model is limited to a bus-based topology, while our analytical estimation model is not restricted to a particular architecture, but it is flexible enough to analyze different kinds of communication architectures.

3. Problem statement and overview of the proposed approach

3.1. Preliminaries

In this section, we define the concepts and key assumptions underlying our approach, and follow-up with a formal problem statement.

Definition 1 (*Application model*). A real-time application is modelled as a task graph (expressed as a Directed Acyclic Graph) $DAG = \{T, L, D\}$, where $T = \{t_1, t_2, \ldots, t_k\}$ is a set of k periodic tasks that must be executed in a certain order to produce the desired results under a real-time constraint (*RTC*), $L = \{l_{12}, l_{13}, \ldots, l_{jk}\}$ represents a set of h unidirectional channels or links that connect the tasks with each other as well as indicate their data dependencies, and $D = \{d_{12}, d_{13}, \ldots, d_{jk}\}$ specifies the amount of shared data associated to each link.

Definition 2 (*MPSoC template*). In this work, we discuss our DSE methodology in a context of MPSoC architectures that are composed of respectively p, m and n component holders for Processing Elements (PEs), Storage Elements (SEs) and Network Elements (NEs), where the connections between different components as well as the architectural topology (defined by the number and type of network elements) are already fixed. More specifically, this paper focuses on MPSoC templates composed of several homogeneous RISC processors completed with a few different hardware dedicated blocks[1], rather than MPSoCs based on various PE types having different and multiple computational characteristics. As a consequence, different *architecture instances* can be derived from the same MPSoC template by varying the number and type of processing elements and storage elements, as well as their allocation in the component holders of the architecture. New templates can be also generated by creating new topologies with different network elements.

Assumption 1. We assume that there is a library of configurable resource models for the processing element, storage element and network elements, which can be instantiated and configured properly to build the architecture templates mentioned in Definition 2. These components and their configuration parameters (such as read/write latency, operating frequency, storage capacity, and network arbitration policy) can also be used by the system designer in both the pruning phase and simulation phase. For example, using such parameters, a processing element can be configured as a

[1] A hardware dedicated block can represent different types of hardware accelerators or co-processors such as GPU, VLIW processor, and ASIC. How hardware dedicated block exploits different degrees of parallelism is not explored in this work, but it can be further conducted in a micro-architectural DSE once a set of optimal solutions have been found using our approach.

generic RISC processor for flexible application support as well as a dedicated hardware block for accelerating a specific task. While setting appropriately the access latency and storage capacity parameters, a storage element can behave as a relatively small but fast cache memory or as a memory bank of large storage capacity and high access time or memory latency.

Definition 3 (*Mapping*). The mapping consists of the process of distributing the application functionality on the available resources of the target architecture. This process addresses three sub-problems: (i) partitioning: refers to the selection of a suitable processing element *type* for each application task, (ii) assignment: decides the type and number of instances for processing elements and storage elements, their locations in the architecture template, as well as which task and channel should be assigned to which component instance (if more than one instance of a component type is present in the architecture), and (iii) scheduling: defines the order of execution for the different tasks assigned to a processing element.

Assumption 2. We assume that all application tasks are assigned to a set of processing elements working in a pipeline fashion, where task duplication and migration are not considered. On the other hand, the inter-task synchronizations and communications can be assigned to (i) a storage element: when two communicating tasks have been assigned to different processing elements, the corresponding inter-task channels should be assigned to a shared storage element accessible by such two processing elements, and therefore a communication time needs to be taken into account, or (ii) inside the processing element: if two communicating tasks (and their associated inter-task channels) reside on the same processing element, it is then reasonable to neglect this communication time, without loss of generality.

Definition 4 (*Communication time*). The time taken for a communication transaction between a processing element and storage element is calculated as the sum of three parts: (i) *protocol delay*: latency associated to communication protocol, (ii) *contention delay*: waiting time due to simultaneous access attempts to shared resources, and (iii) *SE access delay*: time taken to read/write the data from/to a storage element.

Definition 5 (*Computation time*). This is the execution time of an application task on a specific processing element. Thus, all task timing information in the *pt* available types of processing elements (provided by the component library) should be first determined, i.e., each task has a set of computation times $\{wcet(t_i)_1, wcet(t_i)_2, \ldots, wcet(t_i)_{pt}\}$, where $wcet(t_i)_j$ is the execution time of task t_i on the processing element type j. To measure the execution time of a task on a general purpose processor, we use an instruction set simulator. Note that the execution time of a specific task on a dedicated hardware implementation is assumed given, while this hardware block takes an infinite amount of time to execute any of the remaining application tasks (i.e., for those tasks that the hardware block does not implement). As a result, a profile table with *pt* rows and *k* columns can be obtained.

Problem statement. Given a real-time application, an architecture template and a component library, our problem is to find the (sub-)optimal architecture instance(s) for the mapping of the target application. The found solution(s) should satisfy different design constraints and additionally achieve a good trade-off among these design objectives. Each of the objective functions to be optimized is listed below.

- *Max{EPE} = Maximize efficiency in the utilization of processing elements*. The efficiency of a resource usage (epe) determines the fraction of the overall execution time of a processing element during which the resource is busy, i.e., the ratio between the time a processing element is busy and real-time constraint. Therefore, the global efficiency of all assigned processing elements in a design (EPE) is calculated as follows:

$$EPE = \frac{\sum_{i \in NPE} epe_i}{NPE} = \frac{\sum_{i \in NPE} (CMP_i / RTC)}{NPE} \quad (1)$$

where CMP_i is the total computation time due to the tasks assigned to processing element PE_i, and *NPE* is the number of processing elements actually used for mapping the target application on the MPSoC platform. The goal of the optimization consists of exploiting the silicon area as much as possible, and thus avoiding an over-dimensioned design. That is, a design solution is better than others if it achieves higher performance at the same cost/area or the same performance at a lower cost/area.

- *Min{LuB} = Minimize the load unbalancing in processing elements*. Load balancing avoids overloading certain resources which would lead to excessive packet delays in terms of the communication paths and excessive processing delays at computation resources. That is,

$$LuB = \frac{\sum_{i \in NPE} abs(epe_i - EPE)}{NPE} \quad (2)$$

As demonstrated in [19], a load-unbalanced design could also generate additional and unproductive traffic (or access to shared resources) in the communication architecture due to over-synchronization effects. Therefore, this latter also heavily influences the power consumption.

- *Min{IPT} = Minimize the communication traffic*. One way to optimize performance and power consumption of storage elements and network elements consists of minimizing the number of memory accesses and/or minimizing the system load in network elements. Clustering tightly coupled application functions into the same processing resource reduces the accesses in storage elements, and often increases the SoC performance. Moreover, reducing the amount of data exchanged via shared storage elements may also optimize the size of storage elements, and therefore, cost/area of the design. This can be formulated as:

$$IPT = \frac{\sum_{e_{qh} \in VPLs} e_{qh}}{\sum_{d_{ij} \in D} d_{ij}} \quad (3)$$

where the denominator represents the total amount of data exchanged by all tasks of the application, while the numerator indicates the amount of data exchanged between tasks mapped on different processing elements. This latter is denoted as inter-VP links, *VPLs*.

Taking into account the aforementioned mapping problem and the specified design objectives, our multi-objective optimization problem can be formulated formally as follows:

$$(Max\{EPE\}, Min\{LuB\}, Min\{IPT\}) \quad (4)$$

subject to the following design constraints: *RTC* and $(NPE, NSE) \leqslant (p, m)$, where NSE represents the number of storage elements actually used in the mapping solution, while p and m indicate the maximum number of the processing and storage resources available in the target MPSoC platform, as defined in Section 3.

3.2. A two-phase DSE strategy

The overall scheme of our approach is depicted in Fig. 2. Two separated phases can be clearly distinguished: the pruning phase and simulation phase. This way, our two-phase approach enables system designers to exploit (i) the benefits of analytical techniques for a rapid pruning of the large design space, (ii) the accuracy of the simulators for individual design point evaluation, and (iii) the ability to integrate different (or additional) analytical models and simulation tools for system-level design space explorations, aiming to solve the problem of application mapping onto MPSoCs as formulated in this section.

Basically, the first phase uses a set of heuristic-based algorithms to rapidly explore the design space associated to the sub-problems of partitioning, scheduling and assignment. Moreover, an additional analytical model is used during this process to roughly predict the performance of each design point. The goal of this estimation phase is to drastically prune the initial design space, such that only a reduced number of potential mappings is delivered to the next phase. The first phase and its experimental results are the main focus of this paper.

Subsequently, each detected potential mapping together with the application model and the architecture template are combined to generate a system model, which is required and used by a simulator to evaluate more accurately each potential solution. Finally, this simulation process is repeated iteratively until a solution or a set of (sub-)optimal solutions is found.

4. Implementation

In this section, more details about the implementation of distinct intermediate steps of our approach are explained. In particular, we provide in Sections 4.1 and 4.2 the details of the algorithms composing the pruning phase, while the simulation phase is briefly presented in Section 4.3. A detailed description of the simulation phase has been reported in [4,5].

4.1. Heuristic algorithms to produce potential mappings

The goal of the estimation phase is to explore and prune the design space, i.e., identify all potential solutions that meet the design objectives (listed in Section 3), while eliminating those alternatives that do not meet the requirements. To this end, an analytical estimation model and three heuristics-based algorithms have been applied, addressing: (i) HW/SW partitioning, (ii) task clustering, and (iii) cluster assignment. For the sake of illustration, we will use a running example to explain different issues related to these algorithms.

4.1.1. HW/SW partitioning

For a given task graph (*DAG*) and an available set of processing elements, this algorithm selects a suitable type of processing element for each task, i.e., it decides whether a task is executed as software on a processor or by using a hardware block to achieve specific performance requirements. To this end, all tasks are checked formally by the following condition:

$$wcet(t_i)_{proc} \leqslant RTC, \forall t_i \in T \tag{5}$$

Thus, if $wcet(t_i)_{proc}$ is smaller than the real-time constraint, then the task t_i is selected to run on a processor; otherwise, t_i is mapped to be implemented in a specific hardware block. As a result, this partitioning algorithm outputs two kinds of information: (i) *VNT* describes the type of processing element selected for each task

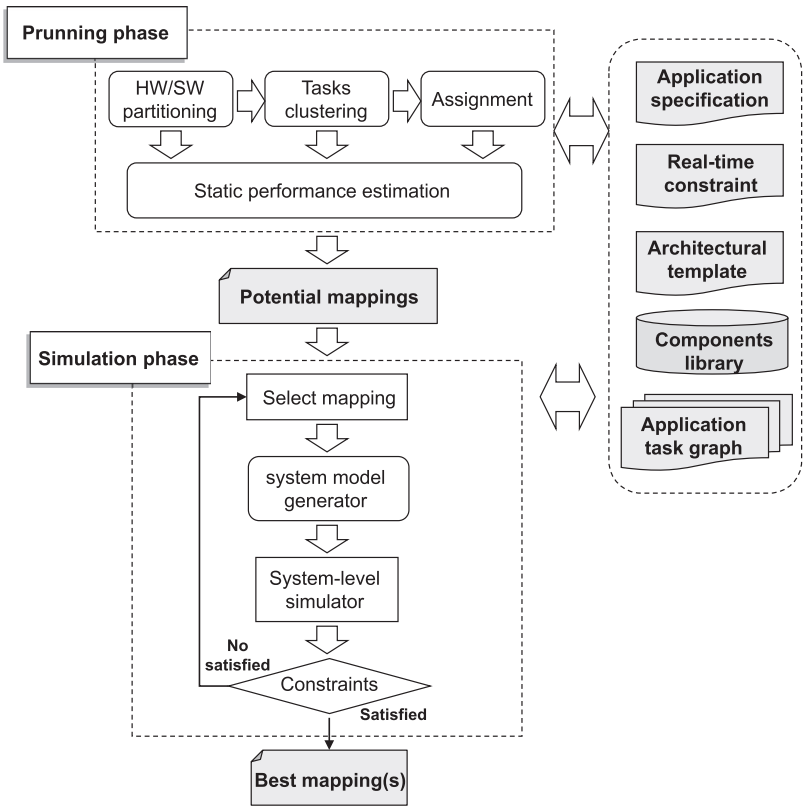


Fig. 2. Overview of our two-phase DSE methodology.

(i.e., hardware dedicated block or generic processor), and (ii) *VCT* specifies the computation time of each task according to its specification in VNT.

For example, Fig. 3a depicts a task graph composed of 9 tasks and 10 channels, where the required timing constraint is 100 time units. According to the *wcet* values listed in Fig. 3b, the HW/SW partitioning algorithm will select the task *C* and *F* for a hardware implementation (*HW1* and *HW2*), while the rest of the tasks will be run on a generic processor. In this example, we suppose that the hardware dedicated blocks can achieve a 10× speedup factor with respect to the SW implementation. Finally, the outputs delivered by our partitioning algorithm are depicted in Fig. 3c. It should be noted that no information about architecture topology/platform is taken into account at this stage.

4.1.2. Task clustering

This step aims at solving the sub-problem of scheduling. To this end, we have used a modified version of the scheduling algorithm presented in [18]. For a given task graph and real-time constraint, the algorithm proposed in [18] enables to schedule the tasks on the minimum number of *Virtual Processors* (*VPs*). A virtual processor is a *logical cluster* composed of several tasks, where each task (inside a virtual processor) is executed according to the order that it was scheduled in the virtual processor.

A key aspect of this scheduling algorithm is that it tends to cluster the tasks sharing large amounts of data in the same virtual processor, while ensuring that the total computation time of each virtual processor is not greater than the real-time constraint, i.e., $RTC > \sum_{\forall t_i \in VP_k} c(t_i), \forall VP$. This way, whenever each virtual processor is assigned to a single physical processing element, this algorithm avoids (i) overloading the system communication architecture with an excessive amount of packets, as well as (ii) introducing large processing delays at computation resources. As a consequence, both the minimization of inter-PE communication traffic and load balancing in processing elements are warranted. More details about this scheduling algorithm can be found in [18].

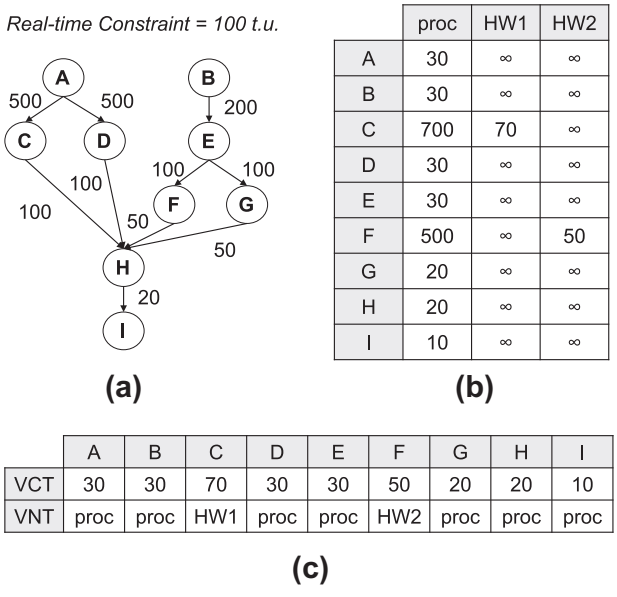
However, this scheduling algorithm only targets architectures with homogenous processing elements, and unlike this paper, it assumes that any task satisfies perfectly Eq. (5), i.e., hardware dedicated blocks are not considered. Thus, we propose a modified version of their scheduling algorithm, where the output of the HW/SW partitioning step is used additionally as input. Basically, our algorithm follows the same aforementioned clustering process to schedule the application tasks, but if a task selected for hardware implementation is found during the scheduling process, such a task is bound exclusively to a new virtual processor. Once all tasks have been scheduled, our algorithm outputs a set of virtual processors $VPs = \{VP_1, VP_2, \ldots, VP_v\}$ and inter-VP links $VPLs = \{vpl_{12}, vpl_{13}, \ldots, vpl_{jv}\}$, where vpl_{ij} represents a link or channel between a task belonging to virtual processor VP_i and another task in virtual processor VP_j, and has an associated communication cost, e_{ij}, which specifies the amount of data sharing for this link. All this information is used as input in the next step, as will be explained later.

Returning to the example shown in Fig. 3, the result of applying the task clustering algorithm is depicted in Fig. 4. Here, four virtual processors and six inter-VP links can be distinguished. Task *C* and *F*, which are selected by the HW/SW partitioning algorithm for hardware implementation, have respectively been assigned to a single logical cluster (virtual processor *VP3* and virtual processor *VP4*). The rest of the tasks are clustered in virtual processors *VP1* and *VP2* for running on generic processors. As mentioned before, each task (inside a virtual processor) is executed according to the order in which it was scheduled. In this example, the execution order of the tasks in *VP1* is *A*, *D* and *B*.

4.1.3. Cluster assignment

4.1.3.1. Virtual processor assignment. The goal of this step is to assign the virtual processors to the processing element holders of the target MPSoC platform, while the assignment of the inter-VP links to the storage elements is carried out in a next step (in Section 4.2). Note that the number of virtual processors provided by the tasks clustering algorithm is v, which can be greater than p (i.e., the number of processing element holders in the MPSoC platform). In our current implementation, we suppose that $p \geqslant v$. Thus, this cluster assignment process implies two design decisions: (i) specify which task (or virtual processor) should be assigned to which particular processing element instance, and (ii) decide the location of each processing element instance in the processing element holder of the MPSoC template.

In order to handle such design decisions, our proposed algorithm firstly assigns the virtual processors to the processing element instances using a one-to-one assignment function, $\Lambda(VP)$, i.e., $\Lambda(VP_i) \neq \Lambda(VP_j), \forall\ VP_i \neq VP_j$ As a consequence, the number of processing element instances to actually instantiate in the MPSoC template corresponds to the number of virtual processors obtained in the task clustering algorithm. Once the v virtual processors have been assigned to v processing element instances, we subsequently generate all possible combinations of location for these processing



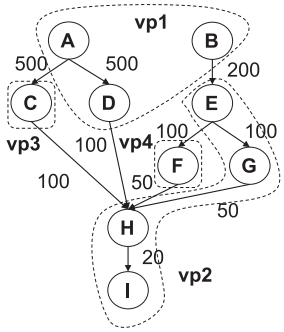
	proc	HW1	HW2
A	30	∞	∞
B	30	∞	∞
C	700	70	∞
D	30	∞	∞
E	30	∞	∞
F	500	∞	50
G	20	∞	∞
H	20	∞	∞
I	10	∞	∞

(b)

	A	B	C	D	E	F	G	H	I
VCT	30	30	70	30	30	50	20	20	10
VNT	proc	proc	HW1	proc	proc	HW2	proc	proc	proc

(c)

Fig. 3. Example of the HW/SW partitioning algorithm. (a) Task Graph and required real-time constraint. (b) WCET for all tasks. (c) VCT and VNT vectors delivered by HW/SW partitioning algorithm.



Number of VP = 4;
VPs={**VP1**(A,D,B), **VP2**(E,G,H,I), **VP3**(C), **VP4**(F)}

Number of VPL=6;
VPLs={$\mathbf{vpl}_{12}$(200), $\mathbf{vpl}_{12}$(100), $\mathbf{vpl}_{13}$(500), $\mathbf{vpl}_{32}$(100), $\mathbf{vpl}_{24}$(100), $\mathbf{vpl}_{42}$(50)}

Fig. 4. An example of the result obtained with the task clustering algorithm.

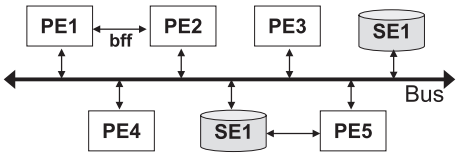
element instances in the target MPSoC template. For example, Fig. 5a illustrates an MPSoC template composed of five processing element holders. If we assign the four virtual processors obtained in our example (Fig. 4) to this template, the total number of allocation alternatives is 5!/(5–4)! = 120.

4.1.3.2. Storage allocation. Although the inter-VP links assignment is not performed in this step, the different combinations of type and allocation for storage elements are indeed explored in this step. Note that this is needed by the analytical estimation model to determine the optimal combination of the number, type and location of storage element instances for the assignment of the inter-VP links. In our example of Fig. 5a, if two storage elements can be instantiated in the MPSoC template and there are two different types of storage elements (i.e., $st = 2$: SDR and DDR), then the resulting number of possibilities for allocating storage elements is $2^2 = 4$.

4.1.3.3. Generation of all PE-SE scenarios for a given topology. All possible scenarios are now generated. For a given MPSoC template, a scenario represents a particular architecture instance and specifies the processing element holder to which each virtual processor is assigned. Moreover, it assigns an available type of storage element to each storage element holder. Note that at this stage inter-VP links are not assigned yet, but this is done at the performance estimation phase, which involves inter-VP links and network elements. Referring to our example again, Fig. 5b shows the output delivered by our cluster assignment algorithm. Considering the above analysis for allocating both processing elements and storage elements in the target MPSoC template, it is evident that the total number of scenarios (or architecture instances) depends directly on the number of virtual processors, the number of processing element holders and the number and types of storage elements. Therefore, the total number of possible scenarios can be approximated as: $(p!/(p-v)!) * st^m = 120 * 4 = 480$.

4.2. Performance estimation for potential mappings

The aforementioned heuristics-based algorithms provide a set of scenarios that tries to optimize the design objectives such as efficiency of resource usage and load balancing in processing



(a)

Scenarios	PE1	PE2	PE3	PE4	PE5	SE1	SE2
SC1	VP1	VP2	VP3	VP4	-	SDR	SDR
SC2	VP1	VP2	VP3	VP4	-	SDR	DDR
…			…			…	
SCk	-	VP4	VP1	VP2	VP3	DDR	DDR
…			…			…	

(b)

Fig. 5. Output delivered by our cluster assignment algorithm. (a) MPSoC used to map the VPs and VPLs obtained in our example. Each PE holder can allocate a processing element of types *proc, HW1 or HW2*, while each SE holder can contain a storage element of types *SDR or DDR*. (b) Generation of the possible scenarios for our example.

elements. Now, this estimation step aims to select analytically those scenarios that satisfy the real-time constraint and minimize the system communication traffic. To this end, we propose an analytical estimation model that includes network element behavior and takes into account both the static and dynamic behavior during the evaluation of each design point. Nevertheless, it should be noted that the goal of our estimation model is to roughly estimate the performance of different design alternatives, and thereby rapidly reducing the large number of potential mapping solutions, since the selected candidates will be evaluated more accurately by a system-level simulator in a later step (i.e., simulation phase).

4.2.1. Analytical estimation model including Network Elements

4.2.1.1. Estimation model. In our approach, the performance of each scenario is measured in terms of total processing element time in order to decide whether the real-time constraint is satisfied, i.e., we check formally the following condition for each assigned processing element of a scenario:

$$\max\{CMP_i + CMM_i\} \leqslant RTC; \quad \forall i = 1..p \tag{6}$$

where CMP_i and CMM_i are the total computation time and total communication time due to the tasks assigned to the processing element PE_i, respectively. This way, the total processing element time is calculated for each assigned processing element of the scenario, and thus, a scenario is considered as a potential solution if the maximum total processing element time (of all total processing element time of a scenario) can meet the above condition.

4.2.1.2. Inter-VP link assignment. In order to determine the communication times for each scenario, the inter-VP links should be first assigned to the storage element instances. At this point, it is important to notice that the location of assigned processing elements in the platform and the communication delays are closely related with each other. That is, when the location of a processing element instance and/or the assignment of a virtual processor changes, the traffic characteristic of the system and the availability of accessible resources change as well. Consequently, the latter should be taken into account during the communication time estimation.

In our case, such availability or accessibility of resources is made explicit by means of the latency or connectivity table, which should be created (manually or automatically) by the system designer for the studied MPSoC template before carrying out the DSE experiments. Basically, a latency table reflects the relationship between the location of processing element holders and the accessibility of the storage element holders. This latter means, for example, when a processing element instance is assigned to a processing element holder in a particular MPSoC template, its corresponding latency table shows all accessible storage element holders and which network elements are available to reach them (from this processing element holder). In addition, it also shows the remaining storage capacity and access latency of the storage element instance allocated in each of the storage element holders. As a result, the latency table can be applied recurrently to any scenario or architecture instance (derived from the target MPSoC template) during the pruning phase.

4.2.1.3. Communication delays. Using such a latency table, we propose a modified version of the estimation model proposed in [18] to calculate the communication delays. Basically, the estimation model in [18] first assigns iteratively the inter-VP links to a set of available storage elements, where priority is given to a link (vpl_{ij}) with the highest e_{ij} value in each iteration. During this process, the communication protocol delay corresponding to each inter-PE link is estimated by means of a set of latency parameters associated to each type of architectural component, which are specified by the

designer. And the storage element access delay is calculated as a product of the amount of shared data (e_{ij}) and the latency of the storage element to which the inter-VP link is assigned. However, the calculation of communication delay in [18] explicitly ignores the contention effects in each communication transaction. Therefore, we introduce a modification to that algorithm in order to solve this issue.

4.2.1.4. NE path-contention delay model. In our approach, once all the inter-VP links have been assigned in their corresponding storage element instances, we identify the network path or network element assigned to each inter-VP link, i.e., the network element used by the processing element instance (PE1) to access a storage element instance (SE1). This process can be done by using the latency table. We first check the available paths to connect both elements (PE1 and SE1). If there is more than one path, we then choose the network element with the lowest number of transactions and inter-VP links assigned to it until that moment. This latter aims at avoiding overloading a particular network element, which may lead to an increase of the communication time due to contention delays.

Once all network paths have been determined, the contention delay can be estimated. In our approach, when several inter-VP links compete to simultaneously use the same shared network element, a communication overhead and contention delay may occur, which is approximated by:

$$ccd = (\alpha - 1) * com(NE_i) \quad (7)$$

where $com(NE_i)$ is the average storage element access delay of all inter-VP links that share the network NE_i, and α is the total number of inter-VP links that (i) share the same NE_i, and (ii) are associated to different processing element sources and/or processing element destinations. Our idea is that the contention delay suffered by an inter-VP link is directly proportional to the average storage element access delay of the inter-VP links competing for the same network element, such that the more inter-VP links (i.e., a high α value) share the same path, the more waiting-time (on average) may be needed to access the data. However, we also would like to note that this way of estimating the contention delays can introduce a certain error or difference with respect to the simulation results when (i) the storage element access delay of inter-VP links (sharing the same network element) present high deviations with respect to the average storage element access delay, and/or (ii) the number of inter-VP links sharing the same network element increases, as will be shown in Section 5.

4.2.2. Encoding format for describing candidate mappings

When all scenarios have been estimated, our algorithm eliminates all those scenarios that do not satisfy the condition (6). As a result, the remaining scenarios are ranked in the ascendant order of the maximum total processing element time, such that a scenario with a small maximum total processing element time is evaluated first in the next simulation-based phase. Therefore, these ranked scenarios are the key to link both pruning and simulation phases in our framework.

We now explain briefly the interface used to describe each scenario (or candidate mapping), since the encoding format is not the main focus of this work. In our framework, such an interface enables to symbolically represent the design space composed by the potential mappings, where each alternative is encoded as two numeric strings (as shown in Fig. 6). The format of these strings is described as follows:

$$\{idPE_1, idPE_2, \ldots, idPE_k; idSE_{12}, idSE_{13}, \ldots, idSE_{jk}\}$$

$$\{tPE_1, tPE_2, \ldots, tPE_p; tSE_1, tSE_2, \ldots, tSE_m\}$$

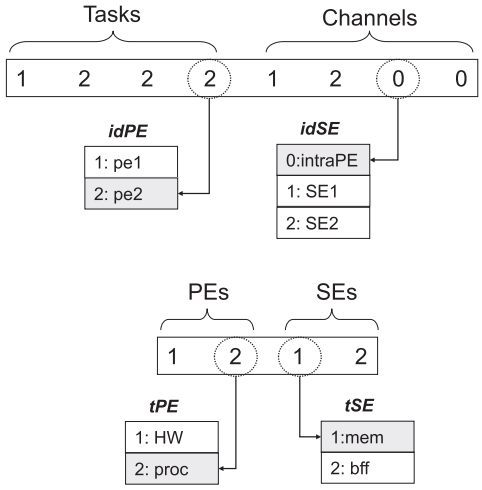


Fig. 6. Example of mapping solution description strings.

where $idPE_k$ indicates that the task t_k is mapped on the processing element holder *idPE*, $idSE_{jk}$ means that the link l_{jk} is mapped on the storage element holder *idSE*, while tPE_p and tSE_m represent the processing element instance type and the storage element instance type allocated in processing element holder p and storage element holder m, respectively.

Note that there exist many other valid encoding formats, and in our case, this choice is mainly due to three reasons. First, it allows us to modularly define a very large design space, where our representation scheme guarantees that each potential solution receives a unique encoding value. Second, both the vector length and the variable values are not fixed, but they are dynamically updated for each experiment according to the user specifications, i.e., this representation scheme has a strong impact on the scalability and flexibility of the framework. And last but not least important, the use of this interface itself allows each phase of our approach to act like an independent black box inside the framework. As a result, distinct methods based on different pruning techniques and/or evaluation mechanisms can be integrated in a plug-and-play fashion.

4.3. Simulation phase

We briefly summarize the simulation phase previously reported in [4,5]. Although different simulators can be plugged in our framework, a system-level simulation tool called CASSE [15] has been used at the moment to more accurately evaluate each potential solution of the pruned design space. CASSE follows the Y-Chart methodology [26,27], covering application and architecture modelling, as well as mapping and analysis within a unified simulation environment. Moreover, this tool enables system designers to configure numerous design variables, which often are not taken into account by analytical models, such as bus arbiter policy, processor scheduler, memory map, different clock domains, size and number of data packets that can simultaneously circulate on the network, and so on.

In the simulation phase, each potential solution (produced by the pruning phase) is simulated in order to find a solution or a set of solutions meeting the design constraints. Note that when no potential mapping can achieve such user constraint, no solution is output by our approach. On the other hand, it should be mentioned that in order to simulate each potential solution in CASSE, a system model (consisting of an application model, architecture model and mapping model) of each design point should be

generated first. As will be explained later, such a system model is generated automatically in our approach taking into account the application source code, components library as well as the mapping strings (provided by the pruning phase). As a consequence, this allows for simulating different design points in a completely automated way without any human intervention.

CASSE currently only has the capability to gather performance information during simulation (e.g., processing time, network load fluctuations, memory accesses, etc.). As a consequence, this limitation of CASSE requires system designers to jointly use (in each DSE experiment) other tools capable of providing other important quantitative metrics such as energy/power and cost/area. The coupling of multiple evaluation tools in our DSE framework is, however, not addressed in this work, but is considered as future work.

Finally, it is important to highlight that besides being a fairly accurate simulator for early DSE, CASSE also ensures deadlock-free task mappings and scheduling for feasible design points, which also makes CASSE particularly suitable for simulations and DSE at system level. Since the CASSE tool is beyond the scope of this paper, the interested reader is referred to [15] for more information about the implementation details, speed-accuracy trade off, the specification method of the system model, etc.

5. Visual tracking case-study: experimental results

In this section, we present several sets of experimental results aiming to compare our two-phase DSE approach with other DSE approaches. To this end, we present a comparative analysis (in terms of run-time and quality of the mapping solutions) between our DSE strategy and other approaches based only on analytical models or simulations, and therefore demonstrating the capabilities and benefits of using our hierarchical approach for the kind of problem addressed in this work.

All these DSE experiments have been carried out using NASA [4,5], which is a single, generic and modular framework for system-level DSE experiments. NASA has three major benefits. First, this highly modular framework uses well-defined interfaces to easily integrate different system-level simulation tools as well as different combinations of search strategies in a simple plug-and-play fashion. Second, NASA deploys a so-called *dimension-oriented DSE approach*, allowing designers to configure the appropriate number of, possibly different, search algorithms to simultaneously co-explore the various design space dimensions. Last, NASA integrates a generator capable of automatically converting an abstract description of a design point (e.g., numeric strings) to the specific system model required by the simulator plugged into NASA. This way, the integration of a new system-level simulator in NASA only requires the adaptation of this module, while all other modules remain unaffected. As a result, NASA provides a flexible and re-usable framework for the systematic exploration of the multi-dimensional design space, starting from a set of relatively simple user specifications. The overview of NASA framework is presented in [4], and more details about the implementation of NASA can be found in [5].

5.1. NASA configurations for Visual Tracking DSE experiments

We now provide details of a particular example or case study. The studied MPSoC template and real-time application are shown in Fig. 7. The latency table associated with such a template and the possible configuration parameters of the different architectural components are depicted in Fig. 8c and d, respectively. In this case, we have selected an MPSoC template consisting of several homogeneous RISC processors completed with a few different hardware dedicated blocks in a bus-based architecture. Basically, this particular MPSoC template may consist of up to 6 processing elements of types ARM-9 or hardware blocks, up to 3 storage elements of either single (SDR) or double (DDR) data rate types, and up to 2 AMBA

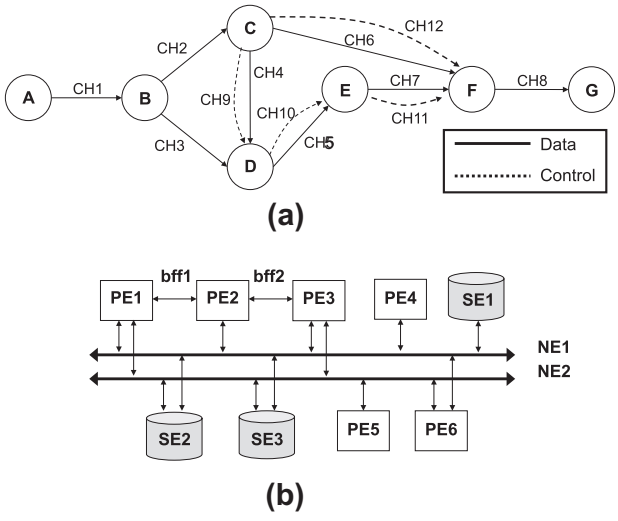


Fig. 7. Application and MPSoC template used in our DSE experiments. (a) Task-graph of the target application model. (b) MPSoC template composed of 6 component holders for PE, 3 component holders for SE, and 2 NE.

Tasks		wcet per frame (ms)
A	Filter	28,63
B	Conversion	17,02
C	Pattern	15,99
D	SAD	355,95
E	Matching	3,41
F	Threshold	1,10
G	Vector	0,50

(a)

Channels (Bytes)			
CH1	19024	CH7	12024
CH2	7000	CH8	2012
CH3	7000	CH9	32
CH4	514	CH10	32
CH5	8224	CH11	32
CH6	15082	CH12	32

(b)

	Bff1 (128B)	Bff2 (128B)	SE1	SE2	SE3
PE1	0	∞	Lat (NE1)	Lat (NE1,NE2)	Lat (NE1,NE2)
PE2	0	0	Lat (NE1)	Lat (NE1)	Lat (NE1)
PE3	∞	0	Lat (NE1)	Lat (NE1,NE2)	Lat (NE1,NE2)
PE4	∞	∞	Lat (NE1)	Lat (NE1)	Lat (NE1)
PE5	∞	∞	∞	Lat (NE2)	Lat (NE2)
PE6	∞	∞	Lat (NE1)	Lat (NE1,NE2)	Lat (NE1,NE2)

(c)

Architecture element	Number	Type	Value
PE	≤ 6	RISC	ARM-9 (Round robin scheduler)
		Accelerator	Dedicated hardware (11x speedup)
SE	≤ 3	SDR	Capacity=256KB; Latency=5
		DDR	Capacity=28KB; Latency=1
NE	≤ 2	AMBA bus	32 bit width, Round robin arbiter, I/O buffered

(d)

Fig. 8. Target application and architecture template parameters. (a) Application tasks. (b) Application channels. (c) Latency table associated to the target MPSoC platform. (d) Architecture parameters of the target MPSoC platform.

busses. Note that although both NASA and CASSE allow designers to explore the design space of configurations of architectural components (e.g., the bandwidth in network elements, the size of storage elements, scheduler policy in processing elements, etc.), micro-architectural DSE is not the scope of this work. Therefore, predefined parameters for component configurations are used in our experiments, as shown in Fig. 8d.

The application that is mapped onto the MPSoC is the visual tracking algorithm presented in [19]. This is a stream-based data processing application, and is characterized by a real-time requirement and high volume of data and control traffic exchanged among different application tasks. Basically, this visual tracking algorithm applies a correlation or block matching technique to continuously track a specific target in the incoming image frames. The block or pattern size and frame size used in our experiments are 24×24 and 320×240, respectively. The task-graph of the application is depicted in Fig. 7a, where the amount of data associated to both control flow (represented by dotted lines) and data flow (shown as black lines) is expressed in bytes (as shown in Fig. 8b), while the computation time of each task (profiled in ADS [28] for the ARM-9 processor) is listed in Fig. 8a.

Considering such user specifications, we have performed three sets of experiments.

- *Estimation*. Only the pruning phase of our hierarchical approach has been used to find the (sub-)optimal mapping(s) in this set of experiments. Since no simulations are needed in this set of experiments, it can be carried out without using CASSE. Note that a single solution (or a set of estimated solutions) is produced in this case.
- *Estimation + Simulation*. We repeat the experiment using our two-phase approach. To this end, we have plugged into NASA our heuristic algorithms, analytical model and CASSE. Evidently, the interfaces used in our hierarchical DSE strategy (depicted in Fig. 2) satisfy the interface requirements of NASA framework.
- *Simulation*. The last set of DSE experiments is based only on simulations (i.e., using CASSE). Rather than simulating exhaustively all design points of the design space, we coupled to CASSE two genetic algorithms (GAs) to guide the searching process and to address our multiobjective optimization problem. That is, we plugged into NASA both CASSE and GAs, and we apply a dimension-oriented DSE methodology [4,5] in order to co-explore the design space (consisting of the architectural components and mapping dimensions).

We use a proprietary implementation of the genetic algorithms in our experiments, but any existing genetic algorithm such as SPEA2 or NSGA-II [16] could also have been used. Note that our interest is not focused on the type of genetic algorithm used in each experiment. Instead, we aim to analyze and compare the behavior of the DSE process guided by the genetic algorithms and by the heuristic algorithms proposed in this work.

Finally, the chromosome of the genetic algorithm, which represents the mapping solution, is defined with the same string-based format shown in Fig. 6. The GA parameter values used in our DSE experiments are listed in Table 1. Note that these parameter values have been selected after experimenting with different combinations of parameters settings, and resulted to be the best for these DSE experiments. Basically, our genetic algorithm can perform a 2-point crossover and can randomly mutate the value of only a single gene per chromosome/individual, where these operators are applied according to their associated probabilities (*pc*: probability of crossover, and *pm*: probability of mutation). Moreover, the fitness functions of the genetic algorithms have been formulated as defined in Section 3.

Table 1
GA parameters used in the third set of experiments.

Parameter	Number	Values
Selector (*S*)	1	Proportional with elitism
Crossover (*C*)	1	2-points
C probability (*pc*)	5	[0.1,0.3,0.5,0.8,1.0]
Mutation (*M*)	1	Single gene mutation per individual
M probability (*pm*)	5	[0.1,0.3,0.5,0.8,1.0]
Population size	10	Nr. of individual per iteration
Iterations	21	–

5.2. Analysis of efficiency and accuracy between different DSE approaches

In order to compare the experimental results obtained with different DSE approaches, we have selected four metrics:

- (M1): Number of explored design points.
- (M2): Total run-time of the DSE experiment.
- (M3): Performance achieved by the (sub-)optimal solution found in the exploration.
- (M4): Number of simulations until finding the first (sub-)optimal solution.

5.2.1. Efficiency of the DSE process

For the target application and MPSoC template depicted in Fig. 7, the size of the design space explored in our case study can be roughly approximated as follows: $6^7 * 11^{12} * 3^2 \approx 7.9 \times 10^{18}$ design points. CASSE requires on average 30 s to simulate a single design point on a PC with a Pentium IV processor at 1.6 GHz (running Linux) and 2 GB main memory. Then, it is evident that an exhaustive evaluation of such a design space with CASSE is infeasible.

The results obtained with the aforementioned DSE experiments are summarized in Table 2. At a first glance, it can be seen that for different timing constraints (RTC = 25 frames/s and RTC = 28 frames/s, i.e., processing 1250 packets/s and 1400 packets/s, respectively), both the DSE approaches based on only the pruning phase and our two-phase DSE approach can cover a large number of design points, and can achieve a valid solution after a reasonable small number of simulations. On the other hand, the results obtained with the GA-based DSE experiments suggest that genetic algorithms can also find good solutions after exploring only a reduced number of alternatives. However, the genetic algorithms need (on average) 1800 s and 6000 s (i.e., for 60 and 200 simulations) to reach the first solution that meets the real-time constraints of 25 frames/s and 28 frames/s, respectively. That is, comparing with GA-based DSE experiments, our hierarchical DSE approach seems to be capable of improving the efficiency of DSE (in terms of total run-time to reach the first valid (sub-)optimal mapping) by two orders of magnitude.

5.2.2. Accuracy of the analytical model

To determine the accuracy of the proposed analytical model, we use CASSE to simulate the potential mapping solutions generated by the pruning phase, after which we compare the simulated performance with the estimated performance. Such analysis reveals that the accuracy of our estimations (in these DSE experiments) varies from 1.1% to 8.2%, as can be deduced from Table 2. For example, while our estimation model predicts a performance of 1266 packets/s in the case of RTC = 25 frames/s, the simulated performance is 1252 packets/s. Similarly for RTC = 28 frames/s, when our analytical model delivers a performance of 1425 packets/s, the simulation only indicates 1308 packets/s.

The above results outline two aspects of our analytical model. First, the accuracy of our analytical model presents a reasonable

Table 2
Comparison of the DSE efficiency between the three sets of experiments.

DSE approach	RTC = 25 frames/s			
	Nmbr. of explored design points	Total runtime of the experiment (s)	Performance of the optimal solution[a]	Nmbr. of simulations until first optimal solution
Estimation	$16{,}875 \times 10^6$	30	1252 (1266)	–
Estimation + Simulation	$16{,}875 \times 10^6$	30	1252 (1266)	1
Simulation	210	1800	1263 (–)	60
	RTC = 28 frames/s			
Estimation	$253{,}125 \times 10^6$	30	1308 (1425)	–
Estimation + Simulation	$253{,}125 \times 10^6$	90	1401 (1411)	3
Simulation	210	6000	1408 (–)	200

[a] Performance obtained in the simulation (estimation).

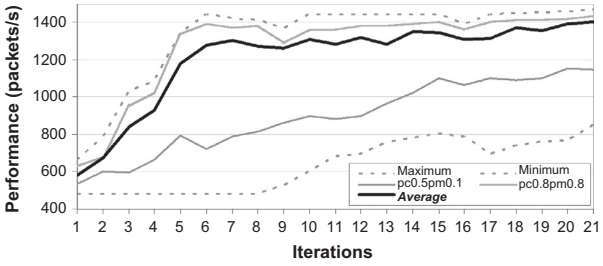


Fig. 9. Results obtained in GA-based DSE experiments.

error margin as compared to the simulations. As already mentioned in the introduction section, such margin is inherent in any analytical estimation model, which emphasizes the need of a simulation-based phase in our framework. More specifically, this variation of accuracy in our estimation model is mainly due to (i) the assumptions adopted in our model for estimating contention delay (i.e., Eq. (7)), and (ii) the lack of other relevant variables that should be considered in the estimation of communication time (e.g., the network arbiter actually used in the system design). As a result, when the number of resources competing for shared resources increases, the occurrence of contention becomes more frequent and complex. Therefore, our analytical model becomes less accurate, increasing the error margin between analytically estimated and simulated results.

Another critical aspect of our analytical model points out that the estimations cannot always satisfy the real-time constraints. In fact, this issue is an immediate consequence of the above explanations, and it is also visible in the results shown in Table 2. For example, in the case of a real-time constraint RTC = 28 frames/s, the estimated performance of the first potential solution (generated by our pruning phase) is 1425 packets/s, while the simulation reveals that the actual performance is under the real-time constraint (i.e., M3 = 1308 < 1400 packets/s). Moreover, three potential solutions (M4 = 3) were simulated before reaching the first valid mapping. Therefore, this latter suggests that the potential solutions provided by our pruning phase may include invalid mappings (i.e., incapable of achieving the required performance). These invalid mappings are identified and discarded in the simulation phase.

5.3. Analysis of the quality of the optimal solutions

Although our proposed approach and GA-based DSE can provide solutions satisfying different user constraints and design objectives, the quality of such solutions can be also radically different. Many metrics can be used in these comparisons. In this paper, we measure the quality of the solution in terms of the different design objectives and constraints defined in Section 3:

- (O 1): Maximize the efficiency of resource usage.
- (O 2): Minimize the load unbalancing between processing elements actually used in the solution.
- (O 3): Minimize the total communication traffic in the system.
- (R 1): Real-time constraint of the target application.
- (R 2): Constraint on the maximum number of processing elements and storage elements that can be used in the mapping solution.

Before analyzing these results, it should be noticed that genetic algorithms are extremely sensitive to their parameters such as the initial population, probability associated to crossover (pc) and mutation (pm) operator, etc. [3,5,16]. This last aspect can be illustrated in Fig. 9, where the gray dotted lines indicate the top and lower boundaries closing the results obtained with 40 different initial populations and combinations of pc and pm. The black line represents the average performance in each iteration reached by the individuals of all GA-based experiments, while the gray lines are two particular examples extracted from this set of experiments. It can be clearly seen that, the experiment labelled *pc0.8pm0.8* (i.e., $pc = 0.8$ and $pm = 0.8$) not only needs fewer simulations to reach a valid solution, but can also converge progressively to higher performance solutions, while the other experiment labelled *pc0.5pm0.1* can hardly reach the minimum real-time constraint after 20 iterations.

Table 3 lists the mapping solutions obtained with our DSE approach, as well as the first (sub-)optimal mapping solution reached in the GA-based experiment *pc0.8pm0.8* for different real-time constraints. More specifically, we indicate the type of processing elements and storage elements allocated in the component holders actually used in each solution, as well as the tasks and channels mapped on each of them. Examining the data in Table 4, it can be seen that for both real-time constraints, our approach can provide solutions using fewer processing elements and storage elements. The number of assigned processing elements also has a direct impact on the system communication traffic: the more processing elements are used, the fewer inter-VP links are assigned inside processing elements, and thereby, increasing the volume of exchanged data via shared resources and lengthening the actual execution time of the tasks.

Note that these results do not mean that genetic algorithms cannot reach the (sub-)optimal mapping (or even other optimal alternatives different than) solutions obtained with our approach. As demonstrated in our previous work [5], the multi-dimensional DSE based on several genetic algorithms can progressively converge towards the optimal solutions of the design space when the number of iterations is increased. However, this implies a

Table 3
Mapping solutions obtained in our DSE experiments.

RTC (frames/s)	Task mapping	Channel mapping
25		
Estimation + Simulation	Proc1 = {A}, HW3 = {D}, Proc2 = {B,C,E,F,G}	Bff1 = {1}, Bff2 = {3,4,5,9,10}
Simulation	Proc1 = {B,C}, Proc3 = {E}, HW5 = {D}, Proc4 = {A,F,G}	SDR1 = {1,6,7,9,11,12}, DDR2 = {3,5,10}, DDR3 = {4}
28		
Estimation + Simulation	Proc1 = {A}, HW3 = {D}, Proc2 = {B}, Proc6 = {C,E,F,G}	Bff1 = {1}, Bff2 = {3}, DDR1 = {2,9,10}, DDR2 = {4,5}
Simulation	Proc1 = {B}, Proc5 = {C}, HW4 = {D}, Proc3 = {F,G}, Proc6 = {A,E}	SDR1 = {3,6,10,11}, DDR2 = {2,7,9}, DDR3 = {1,4,5}

Table 4
Comparison of the quality of the mapping solutions.

	RTC	Nmbr of used resources[a]	Resource usage efficiency (%)	Load unbalancing (%)	Communication traffic (%)
Estimation + Simulation	25	3 (0)	82.507	8.362	49.045
Simulation		4 (3)	61.881	26.678	87.308
Estimation + Simulation	28	4 (2)	69.307	16.078	58.903
Simulation		5 (3)	55.446	27.771	97.124

[a] Number of PE (SE) used in the solution.

higher total run-time, therefore highlighting even more the potential efficiency achieved with our hierarchical approach.

Another alternative is to replace the heuristic algorithms used in our pruning phase by a genetic algorithm, since both our heuristic algorithms and a genetic algorithm (or any other approximated exploring strategy) visit only a limited number of design points to provide a set of near-to optimal solutions. However, the variation of the results achieved in experiments with different GA parameter configurations, or even between different GA executions due to the GA's stochastic behavior, can represent an inconvenience for using a genetic algorithm in the pruning phase. As shown in Fig. 9, the margin of variation between the maximum and minimum can be up to 65% (measured in the sixth iteration). In our point of view, these results could suggest that, in order to find out the well-tuned configuration of the GA parameters for each studied case, the system designers should repeat the experiments several times with different combinations of GA parameters settings, therefore decreasing the attractiveness (in terms of time and effort) of using genetic algorithms in DSE experiments.

6. Conclusions and future work

In this work, we have focused on the problem of mapping a real-time application on a MPSoC architecture. More specifically, we have presented a two-phase DSE strategy and we have elaborated a set of analytical methods deployed in our approach. Basically, the first phase prunes the design space by using a set of analytical models. Our objective is to reduce the number of potential solutions that should be accurately simulated in the second phase, in order to find the (sub-)optimal mapping solution(s) satisfying the user constraints and design objectives.

The *proof-of-concept* DSE experiments presented in this paper are aimed to demonstrate and illustrate the key properties of our framework for a target application and a particular MPSoC platform template. The results obtained in our experiments are analyzed in terms of both efficiency of the DSE process and quality of the optimal solutions found in the DSE process. Compared to the traditional DSE based on simulations guided by a genetic algorithm, the promising experimental results of our approach reveal a potential efficiency increase of two orders of magnitude. Regarding quality, our approach seems also to be capable of reaching mapping solutions with a significant quality increase, i.e., solutions satisfying real-time constraints while optimizing other design objectives such as resource usage efficiency, system traffic load and processor load balancing.

Our future work is oriented toward three objectives. First, we plan to carry out more experiments using more applications (e.g., large real and/or synthetic task graphs) and a wide range of architectures. This will lead to validate the suitability and scalability of our approach for exploring larger and more complex MPSoC design spaces. Second, we intend to demonstrate the flexibility of our overall framework to integrate different kinds of pruning/exploration algorithms and evaluation methods. To this end, we are currently deploying new case studies by using a trace-driven simulator called Sesame [16] along with the PISA optimization framework [25]. Finally, we would like to point out that currently our analytical models do not directly optimize design objectives such as power/energy and/or cost/area. This is a limitation of our approach, since both power/energy and cost/area metrics are key issues in the design and optimization problems of any modern MPSoC. Therefore, we aim to extend our analytical models to further trade off these non-linear and contradictory design objectives in our DSE strategy.

Acknowledgements

This work was supported in part by Spanish Ministry for Science and Technology ref. TEC2009-14672-C02-02, and co-funded by European Social Fund and Canary Agency for Research and Innovation.

References

[1] G. Martin, Overview of the MPSoC design challenge, in: 43th Design Automation Conference (DAC), 2006, pp. 274–279.
[2] M. Gries, Methods for evaluating and covering the design space during early design development, Integration, the VLSI Journal 38 (2) (2004) 131–183.
[3] S. Künzli, L. Thiele, E. Zitzler, A modular design space exploration framework for embedded systems, IEE Proceedings – Computer and Digital Techniques 152 (2) (2005) 183–192.
[4] Z.J. Jia, A.D. Pimentel, M. Thompson, T. Bautista, A. Núñez, NASA: A generic infrastructure for system-level MP-SoC design space exploration, in: 8th IEEE Workshop on Embedded Systems for Real-time Multimedia (ESTIMedia), 2010, pp. 41–50.
[5] Z.J. Jia, A.D. Pimentel, M. Thompson, T. Bautista, A. Núñez, A system-level infrastructure for multi-dimensional MP-SoC design space co-exploration, ACM Transactions on Embedded Computing Systems 2013 (in press).
[6] P. Van Stralen, A.D. Pimentel, Scenario-based design space exploration of MPSoCs, in: IEEE International Conference on Computer Design (ICCD), 2010, pp. 305–312.
[7] K. Lahiri, A. Raghunathan, S. Dey, Efficient exploration of the SoC communication architecture design space, in: International Conference on Computer Aided Design (ICCAD), 2000, pp. 424–430.
[8] S. Mohanty, V. Prasanna, S. Neema, J. Davis, Rapid design space exploration of heterogeneous embedded systems using symbolic search and multi-granular simulation, in: Workshop on Languages, Compilers and Tools for Embedded

Systems: Software and Compilers for Embedded Systems (LCTES-SCOPES), 2002, pp. 18–27.
[9] G. Ascia, V. Catania, A.G. Di Nuovo, M. Palesi, D. Patti, Performance evaluation of efficient multi-objective evolutionary algorithms for design space exploration of embedded computer systems, Applied Soft Computing 11 (1) (2011) 382–398.
[10] G. Palermo, C. Silvano, V. Zaccaria, ReSPIR: a response surface-based pareto iterative refinement for application-specific design space exploration, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD) 28 (12) (2009) 1816–1829.
[11] S. Eyerman, L. Eeckhout, K. De Bosschere, Efficient design space exploration of high performance embedded out-of-order processors, in: Design Automation and Test in Europe (DATE), 2006, pp. 1–6.
[12] H. Javaid, A. Ignjatovic, S. Parameswaran, Rapid design space exploration of application specific heterogeneous pipelined multiprocessor systems, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD) 29 (11) (2010) 1777–1789.
[13] J. Madsen, T. Stidsen, P. Kjarulf, S. Mahadevan, Multi-objective design space exploration of embedded system platforms, in: IFIP Working Conference on Distributed and Parallel Embedded Systems (DIPES), 2006, pp. 185–194.
[14] C. Lee, S. Kim, S. Ha, A systematic design space exploration of MPSoC based on synchronous data flow specification, Journal of Signal Processing System 58 (2) (2010) 193–213.
[15] V. Reyes, Methods and tools for the design of Multimedia MPSoC platforms at system level, Ph.D. thesis, University of Las Palmas de Gran Canaria, Spain, 2008.
[16] C. Erbas, System-level Modelling and Design Space Exploration for Multiprocessor Embedded System-on-Chip Architectures, Ph.D. thesis, University of Amsterdam, the Netherlands, 2007.
[17] J. Eker, J. Janneck, E.A. Lee, J. Liu, X. Liu, J. Ludvig, S. Neuendorfeer, S. Sachs, Y. Xiong, Taming heterogeneity – the ptolemy approach, Proceedings of the IEEE 91 (1) (2003) 127–144.
[18] Z.J. Jia, T. Bautista, A. Núñez, Real-time application to multiprocessor-system-on-chip mapping strategy for a system-level design tool, IEE Electronic Letters 45 (12) (2009) 613–615.
[19] Z.J. Jia, T. Bautista, A. Núñez, C. Guerra, M. Hernández, Design space exploration and performance analysis or the modular design of CVS in a heterogeneous MPSoC, in: Conference on Reconfigurable Computing and FPGA (ReConFig), 2008, pp. 193–198.
[20] S. Künzli, F. Poetti, L. Benini, L. Thiele, Combining simulation and formal methods for system-level performance analysis, in: Design Automation and Test in Europe (DATE), 2006, pp. 236–241.
[21] C. Erbas, A.D. Pimentel, M. Thompson, S. Polstra, A framework for system-level modelling and simulation of embedded systems architectures, EURASIP Journal on Embedded Systems 1 (2007) 2–20.
[22] M.R. Garey, D.S. Johnson, Computers and Intractability: A Guide to the Theory of NP-Completeness, W. H. Freeman & Co., 1990.
[23] M. Kim, S. Banerjee, N. Dutt, N. Venkatasubramaian, Design space exploration of real-time multi-media MP-SoCs with heterogeneous scheduling policies, in: 4th International Conference on Hardware/Software Codesign and System, synthesis (CODES+ISSS), 2006, pp. 16–21.
[24] L. Thiele, I. Bacivarov, W. Haid, K. Huang, Mapping applications to tiled multiprocessor embedded systems, in: 7th International Conference on Application of Concurrency to System Design (ACSD), 2007, pp. 29–40.
[25] PISA, <www.tik.ee.ethz.ch/sop/pisa/> (last accessed 01.12.12).
[26] K. Keutzer, S. Malik, A. Newton, J. Rabaey, A. Sangiovanni-Vincentelli, System level design: orthogonalization of concerns and platform-based design, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 19 (12) (2000) 1523–1543.
[27] B. Kienhuis, E. Deprettere, K. Vissers, P. Van Der Wolf, An approach for quantitative analysis of application-specific dataflow architectures, in: IEEE International Conference on Application-Specific Systems, Architectures and Processors, 1997, pp. 338–349.
[28] ARM Developer Suite, Version 1.2. <www.arm.com> (last accessed 01.12.12).

Zai Jian Jia is a post-doctoral researcher in Institute for Applied Microelectronics, at University of Las Palmas de Gran Canaria, Spain. He received the MSc (2006) and Ph.D. (2011) degrees from the School of Telecommunication Engineering at University of Las Palmas de Gran Canaria. He was a Visiting Researcher in the Computer Systems Architecture group at University of Amsterdam (2009), and in the Center for Embedded Computer Systems at University of California, Irvine (2010). His research interests include methods and tools for mapping applications on multiprocessor embedded systems, design space exploration strategies at system level, analysis of real-time embedded systems and, image and video processing.

Antonio Nunez is Full Professor of Electronic Engineering at the University of Las Palmas de Gran Canaria, Spain. He received the engineering (1974) and Ph.D. (1981) degrees from the School of Telecommunication Engineering at the Technical University of Madrid UPM. His current research fields include hardware-software co-design for embedded systems, system-level design of MPSoCs, multimedia processor architectures, and low power optimization of integrated circuits. He is in the steering committees of ISQED, Euromicro DSD, PATMOS and DCIS. He is also associate editor of Elsevier Journal of MICPRO (Embedded Hardware Design), Elsevier Journal of Signal Processing, Springer Journal of Real Time Image Processing, and Springer Journal of Embedded Systems.

Tomás Bautista is Associate Professor in the University of Las Palmas de Gran Canaria, Spain. He received the engineering and Ph.D. degrees from the School of Telecommunication Engineering of the University of Las Palmas de Gran Canaria in 1992 and 1999 respectively. He was a Visiting Scientist with the Fraunhofer Institute for Microelectronic Circuits and Systems in Duisburg, Germany, in 1990 and with the Philips Research Laboratories in Eindhoven, the Netherlands, in 2000. His research fields include synthesis-based design for SoCs and MEMs.

Andy D. Pimentel is associate professor in the Computer Systems Architecture group at the University of Amsterdam. He holds the MSc (1993) and Ph.D. (1998) degrees in computer science, both from the University of Amsterdam. His research interests include system-level design of MPSoCs, design space exploration, performance and power analysis, embedded systems, and parallel computing. He serves on the editorial boards of Elsevier's Simulation Modelling Practice and Theory as well as Springer's Journal of Signal Processing Systems. Moreover, he serves on the organizational committees for a range of leading conferences, such as DAC, DATE, CODES + ISSS, ICCAD, FPL, SAMOS, and ESTIMedia.