IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received December 11, 2017, accepted January 16, 2018, date of publication January 30, 2018, date of current version March 28, 2018.

Digital Object Identifier 10.1109/ACCESS.2018.2799852

R-Codesign: Codesign Methodology for Real-Time Reconfigurable Embedded Systems Under Energy Constraints

INES GHRIBI[1,2,3], RIADH BEN ABDALLAH[2,4], MOHAMED KHALGUI[1,2], ZHIWU LI[5,6], (Fellow, IEEE), KHALID ALNOWIBET[7], AND MARCO PLATZNER[8]

[1]School of Electrical and Information Engineering, Jinan University, Zhuhai 519070, China
[2]National Institute of Applied Sciences and Technology, University of Carthage, Tunis 1080, Tunisia
[3]Faculty of Mathematical, Physical and Natural Sciences, University of Tunis-El Manar, Tunis 2092, Tunisia
[4]Higher Institute of Applied Sciences and Technology, 7050 Mateur, Tunisia
[5]Institute of Systems Engineering, Macau University of Science and Technology, Taipa 999078, Macau
[6]School of Electro-Mechanical Engineering, Xidian University, Xi'an 710071, China
[7]Department of Statistics and Operations Research, King Saud University, Riyadh 11451, Saudi Arabia
[8]Department of Computer Science, University of Paderborn, 33098 Paderborn, Germany

Corresponding authors: Mohamed Khalgui (khalgui.mohamed@gmail.com) and Zhiwu Li (systemscontrol@gmail.com)

This work was supported in part by the Science and Technology Development Fund under Grant 078/2015/A3 and in part by the Deanship of Scientific Research at King Saud University under GrantRGP-1436-040.

ABSTRACT Hardware/software codesign involves various design problems, including system specification, design space exploration, hardware/software co-verification, and system synthesis. An effective codesign process requires accurately predicting the performance, cost and power consequence of any design trade-off in algorithms and hardware configuration. This paper presents a new co-design methodology called R-codesign. Based on new modeling and partitioning techniques for reconfigurable embedded systems, R-codesign creates a task allocation of SW functions and HW behaviors based on the user constraints and using heuristics. The modeling approach relies basically on probabilistic estimations of the executions of system tasks. Hardware and software specifications are the inputs of R-codesign which constructs partitions (clusters of tasks) and maps them to a specified heterogeneous multiprocessor system-on-chip execution platform with field-programmable gate array. Several design constraints are evaluated and tested during the partitioning and mapping process. We have developed a visual environment called SPEX that implements this methodology. SPEX computes a control matrix which is a pre-computation of validated mappings that will occur in a case of a system reconfiguration. SPEX is an open source, fast and provides efficient results for the codesign of reconfigurable embedded systems.

INDEX TERMS Embedded system, reconfiguration, real-time, co-design, MPSoC, FPGA.

I. INTRODUCTION

The complexity of designing embedded systems is constantly increasing which motivates the need for using more efficient tools and design methodologies. Methodologies that employ modeling techniques at a low level abstraction are no more applicable due to this complexity. We propose in this work a new codesign methodology, based on high level abstraction modeling techniques, which deal with coarse grain components that increase productivity. We remember that hardware/software co-design is the technique of designing concurrent hardware and software components of an embedded system. Generally, hardware/software co-design starts with a specification step followed by a modeling step in which designers have to decide which part of the system should be mapped on hardware and which part on software. The hardware/software partitioning step follows. It is a combinational optimization problem that assigns the system functions to the target architecture on the software and hardware domain under the condition of meeting the design constraints. This is a key task in the system level design since the decisions made during this step directly impact the performance and cost of the final implementation. Another aspect of hardware/

software design methodologies is their incapability to roll back hardware/software partitioning decisions. This flexibility is an important aspect allowing to early discover the consequences of a particular hardware/software partitioning decision and, if deemed inappropriate, exploring another [2]. One way of achieving this goal is to develop abstract hardware/software models during the partitioning process which can be used to assess these decisions.

Another main performance issues in embedded systems design is to guarantee the results within a given time. Such systems that have to fulfill posed constraints are called real-time systems [3]. Most of these real-time embedded systems interact with the external environment, which means that task executions are triggered by external events [4]. The system response should be modulated according to the stimulus from outside. Designers of such systems make use of reconfigurable components and the system implementation becomes a kind of building blocks game [5], [6]. These systems undergo unpredictable events that require adequate online decisions so as to maintain the desired performance and schedulability [7]. A reconfiguration event is defined as an internal/external event that leads to add/remove tasks [8]. Consequently, any reconfiguration scenario may increase the energy consumption and/or make some tasks to violate their deadlines. Thus, this flexibility adds more complexity in their design process [9], [10]. Moreover, in real-time systems, tasks scheduling is critical and depends on previous mapping steps among the system processing elements [11], [12].

In the present paper, we propose a methodology called R-codesign for reconfigurable co-design. It aims to find solutions for modeling and partitioning probabilistic real-time systems having multiple reconfiguration scenarios. R-codesign creates a task allocation of SW functions and HW behaviors based on the user constraints and using heuristics. Guaranteed available resources, feasible scheduling and a generation of a reconfiguration controller are the main concern. The gain of the methodology resides in two aspects: (i) the estimation of the execution flow allows to map the most probabilistic functions to be executed, to be stored together, and hence the communication costs will be reduced and the overall performance will be enhanced, (ii) the precomputed mapping of the possible execution scenarios allows to reconfigure the system at run-time with a minimum reconfiguration overhead.

Firstly, R-codesign presents an abstract model for hardware/software systems allowing early exploration of hardware/software executions and evaluation of design alternatives. This model supports incremental refinement and evaluation at multiple abstraction levels. The separation between software and hardware tasks is supposed to be manually done by users. The decision is made by an expert after a complexity study of each module (node in the DAG) who knows the computational requirements of the system processing flow. Hardware is limited to specifically designed tasks that are, taken independently, very simple. Software implements algorithms that allow to complete much more complex tasks. The entry point for R-codesign is a hardware/software specification modeled by a DAG (Directed Acyclic Graph) where nodes are software functions or hardware behaviors. The edges of these DAGs are valued with a probabilistic estimation of their connecting nodes execution along with the communication cost of the communicating nodes. The goal of the methodology is to partition and map all predefined possible configuration scenarios off-line into a hardware target architecture that is mainly an MPSoC and implement a controller that will supervise and reconfigure the system on-line [13]. All the important and more likely configuration scenarios are pre-computed and given as input to the methodology. Each possible configuration is composed of a set of periodic tasks modeled according to the proposed DAGs presentation. Thus, we developed adequate partitioning and mapping techniques for the proposed hardware/software model. This partitioning/mapping approach is called I-codesign. Several design constraints are considered in this work such as the inclusion/exclusion constraint which is related to the functional specification of processors. An optimization phase is applied at the end of the I-codesign using the Kernighan-Lin algorithm [14] in attempt to find an optimal series of interchange operations between communicating elements in the DAGs. I-codesign treats the software functions and the hardware behaviors separately and then a co-simulation step decides whether or not the mapping results satisfy the design constraints. If I-codesign fails to map the specification, then R-codesign reports the issues to the user in order to tune the input parameters. Otherwise, the results are stored in a controller matrix which will be used by the reconfiguration controller at run-time. In case of large systems, the global mapping matrix could be broken-down into small matrices controlled by multiple controllers in a distributed fashion which avoids reconfiguration fetch overhead.

At last, R-codesign is a codesign methodology that allows to rapidly evaluate hardware/software systems using abstract models. The partitioning and mapping results reveal its efficiency. Indeed, it guarantees a feasible solution and enhances the overall performance compared with existing methodologies.

The paper proceeds as follows. The next section describes useful background. Section III presents the system formalization and the notations used in this paper. In Section IV the R-codesign methodology is developed. Section V exposes the experimental results and finally we conclude this paper in Section VI.

II. STATE OF THE ART

Hardware/software codesign can be considered as the process of concurrent and coordinated design of an electronic system comprising hardware as well as software components based on a system description that is implementation-independent [15], [16]. One of the key problems in hardware/software codesign is hardware/software partition-

ing [17]. One of the most relevant works dealing with partitioning is presented in [18]: A very sophisticated integer linear programming model for the joint partitioning and scheduling problem for a wide range of target architectures. This integer program is part of a 2-phase heuristic optimization scheme which aims at gaining better timing estimates using repeated scheduling phases, and using the estimations in the partitioning phases. The work in [19] presents a method for allocation of hardware/software resources for optimal partitioning. During the allocation algorithm, an estimated hardware/software partition is also built. The algorithm for this is basically a greedy algorithm: It takes the components one by one, and allocates the most critical building block of the current component to hardware. The study in [20] shows an algorithm to solve the joint problem of partitioning and scheduling. It consists of basically two local search heuristics: one for partitioning and one for scheduling. The two algorithms operate on the same graph, at the same time. The work in [21] considers partitioning in the design of ASIPs (application-specific integrated processors). It presents a formal frame work and proposes a partitioning algorithm based on branch and bound. The research in [22] presents an approach that is largely orthogonal to other partitioning methods: it deals with the problem of hierarchically matching tasks to resources. It also shows a method for weighting partially defined user preferences, which can be very useful for multiple-objective optimization problems [23].

Along with the partitioning and mapping problem, co-simulation becomes an important area of research for the early validation of design decisions [24], [25]. In co-simulation, the execution of software on CPUs is simulated using a virtual model of the processor hardware or with simulation models as ISS (Instruction Set Simulator). ISS reduces the complexity of the system design compared with performing a pure gate level or register transfer level (RTL) hardware simulation, which is typically too much slow. The co-simulation problem lies in coupling different models to make the hardware simulation sufficiently accurate [26]. Nowadays, we already live in a third generation of co-design technology with cross-level design environments for the synthesis of complex electronic systems [27], [28]. During the last decade, many important milestones of progress with respect to the initial findings have been achieved [29], [30]. Several design environments have been developed: SARA [31], ADAS [32], PTOLEMY [33], and PML [34].

In the present work, we introduce a new co-design methodology based on constructive and iterative partitioning phases. The originality of this work compared with the previously proposed approaches in the literature resides in multiple aspects including:

- A probabilistic estimation of the software models aiming to predict the execution flow which leads to an improvement in the codesign results and a noticeable enhancement in the performance of the system,
- A codesign methodology based on multiple constraints and feasibility analysis that shows good performance enhancements especially in terms of execution time and communication cost,
- A visual tool that implements the methodology and generates the controller table providing tasks mapping to anticipate all reconfiguration scenarios,

III. FORMALIZATION

This section presents the formalization of a hardware/software system specification. We also explain the partitioning techniques used in R-codesign.

A. SYSTEM MODEL

In this study we target an MPSoC hardware architecture as an executing platform [35]. It is composed of a single master tile controlling multiple slave tiles. Each tile is composed of a CPU (Central Processing Unit), a local memory and a reconfigurable hardware allowing the implementation of custom hardware used for acceleration purpose. The master tile includes I/O interfaces. In a classic use case, the master tile receives through these interfaces external data events from sensors and reconfigure the system tasks and hardware behaviors accordingly. The tiles communicate through a communication medium which can be a bus, a NoC (Network on Chip), a crossbar or a shared memory. A hybrid memory model is adopted i.e., each tile has its own private memory. The considered tiles can communicate through a global shared memory where an upper bound on the time required to access the shared resource is considered. We assume that software tasks are those executed by programmable processors (e.g., GPP (General Purpose Processor) and DSP (Digital Signal Processor)).

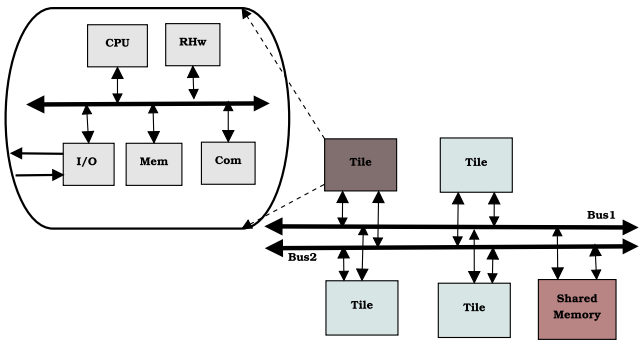


FIGURE 1. Proposed Hardware Model.

Their implementation is an executable code while hardware tasks are those implemented by a specific programmable integrated circuit. Hardware tasks are generally provided as IPs (Intellectual Property) written in a hardware description language (e.g., VHDL, verilog) and implemented by FPGAs or dedicated ASICs (Application Specific Integrated Circuit). We assume that all processors are homogeneous and same for FPGAs in order to simplify the understanding of our methodology through the exposed examples. The proposed

methodology remains valid even with heterogeneous processing elements. Figure 1 presents an example of the hardware model. A tile Hw_i, $i \in [1..\mathcal{L}]$, is characterized by the quadruplet (S_{CPU}, S_{FPGA}, Pw_{FPGA}, Pw_{CPU}, $Freq$), where (i) S_{CPU} is the available memory size in case of CPU, (ii) S_{FPGA} is the FPGA area in term of gates number, (iii) Pw_{FPGA} is the produced electrical power of an PFGA, and (iv) Pw_{CPU} is the produced electrical power of a CPU, and (v) $Freq$ is the range of the available operating frequency. The Bandwidth $BW_{i,j}$ is defined in case of two communicating tiles Hw_i and Hw_j. The memory and power parameters are common to all tiles.

The system model is divided into γ configurations. A configuration ζ_l, $l \in [1..\gamma]$, is a set of tasks to be executed when the configuration is initiated. Each task in a configuration is considered as a graph of elementary functions/behaviors with their intrinsic proprieties and constraints. A task $T_i \in \zeta_l$, $i \in [1..\mathcal{R}]$ is represented by a directed acyclic graph $T_i = (V_i, E_i)$, where (i) V_i is a set of nodes that correspond to behaviors or functions, and (ii) E_i is a set of arcs which describe the connections between functions/behaviors. Each task T_i is composed of $n_{i,1}$ behaviors and $n_{i,2}$ functions. A hardware behavior denoted by $B_{j,i}$ is described as a 5-tuple $B_{j,i} = (E_j^{hw}, M_j^{hw}, C_j^{hw}, D_j^{hw}, P_j^{hw})$, where E_j^{hw} represents the execution time of the hardware behavior on FPGA, M_j^{hw} stands for the number of gates necessary for implementing $B_{j,i}$, C_j^{hw} denotes the power consumption of the hardware behavior, and D_j^{hw} and P_j^{hw} are respectively its relative deadline and period. A software function $F_{k,i}$ is described as a 5-tuple $F_{k,i} = (E_k^{sw}, M_k^{sw}, C_k^{sw}, D_k^{sw}, P_k^{sw})$, where E_k^{sw} stands for the execution time of $F_{k,i}$ on CPU, M_k^{sw} denotes the memory size in byte required by $F_{k,i}$, C_k^{sw} represents the power consumption of the software function, and D_k^{sw} and P_k^{sw} denote respectively its relative deadline and period.

We also consider inclusion/exclusion constraints. They are used to impose at a couple of functions and/or behaviors to be executed either on the same computing unit or on different ones. These constraints are expressed by the following functions:

- **Exclu**($F_{k,i}$) is a set that groups functions which have not to be executed on the same processor with the function $F_{k,i}$. This constraint is modeled within the task representation by marking the symbol $\not\subset$ on the function $F_{k,i}$ which means that $F_{k,i}$ should not be executed with its predecessors on the same computing unit.
- **Inclu**($F_{k,i}$) is a set that groups functions which have to be executed on the same processor with $F_{k,i}$. This constraint is modeled by marking the symbol $\subset$ on $F_{k,i}$ which means that $F_{k,i}$ should be executed with its predecessors on the same computing unit.

The edges are weighted with a couple $\prec Pr_k, Cc_k \succ$ where Pr_k is the probability of executing this edge and Cc_k is the communication cost of data transfer between the two nodes. We consider that the data are always fetched by software functions and propagated to the hardware behaviors where it is attached to. Figure 2 presents an example of the proposed

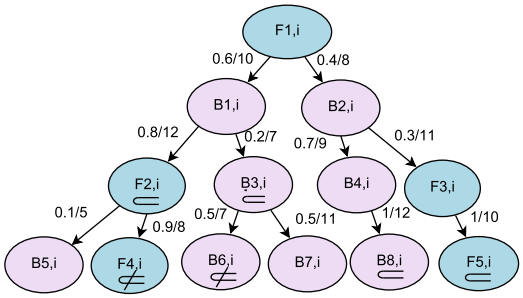


FIGURE 2. A task example.

specification. The system is composed of $n_{i,1} = 8$ hardware behaviors and $n_{i,2} = 5$ software functions connected with valued edges. Inclusion constraints are visible on $F_{2,i}$, $F_{5,i}$, $B_{3,i}$ and $B_{8,i}$ while the exclusion constraints are present on $F_{4,i}$ and $B_{6,i}$. The design constraints are user-defined parameters in the system model which are set according to a prior performance study. The goal of the current paper is to provide a solution that places the tasks in the related devices under real-time, energy and memory constraints.

The aim of the design formalization is to generate a controller allowing an efficient system reconfiguration.

Problem Statement: Given target hardware following the previously described tile based execution model and our R-codesign System model (DAG of tasks comprising software functions and hardware behaviors with a set of constraints along with their execution estimated probabilities), find an appropriate mapping for each possible configuration that respects available hardware resources while satisfying real time and energy constraints.

B. I-CODESIGN METHODOLOGY

The R-codesign partitioning reuses I-codesign partitioning algorithms [36] and extends them to hardware behaviors. The goal of I-codesign is to achieve a concurrent design between the probabilistic task model and the hardware architecture previously described in a manner that fulfills all the system requirements and respects the design constraints. Each phase of the I-codesign has its own constraint(s) and terminates when all the nodes characterized with the specified constraint(s) are mapped. Firstly, we apply I-codesign to software functions ignoring the hardware behaviors. Then in a second step we perform the same steps to hardware behaviors.

1) FUNCTIONAL PARTITIONING

Evaluates the inclusion/exclusion constraints between graph nodes. Then, this phase creates clusters depending on these constraints. The inclusion/exclusion constraint decides the number of the physical components when placing the functions/behaviors.

We place the rest of the functions/behaviors in the already used physical components within the limits of available resources. In the case of resource shortage, other physical components are allocated. If there are available memory and energy on a created cluster, then sub-tasks from different

Algorithm 1 Functional Partitioning Algorithm

```
1: procedure Functional-partition(ttask: tab-task, var tc: tab-Cluster, var tf: tab-function)
2:     for i = 1 to length(ttask) do
3:         for j = 1 to length(ttask[i]) do
4:             if inclusion(ttask[i][j]) then
5:                 Cluster_T(Cluster_1, ttask[i][j], Pred(ttask[i][j]))
6:                 if OK_Memory(cluster1)& OK_Energy(cluster_1) then
7:                     Add_C(cluster_1, tc)
8:                     empty(cluster_1)
9:                 end if
10:            else if exclusion(ttask[i][j]) then
11:                Cluster(cluster_1, ttask[i][j])
12:                Cluster(cluster_2, Pred(ttask[i][j]))
13:                if OK_Memory(cluster_1) & OK_Energy(cluster_1) then
14:                    Add_C(cluster_1, tc)
15:                    empty(cluster_1)
16:                end if
17:                if OK_Memory(cluster_2) & OK_Energy(cluster_2) then
18:                    Add_C(cluster_2, tc)
19:                    empty(cluster_2)
20:                end if
21:            else Add_F(ttask[i][j], tf)
22:            end if
23:        end for
24:    end for
25:    if cluster_1 ≠ 0 then ‘
26:        Add_C(cluster_1, tc)
27:    end if
28:    if cluster_2 ≠ 0 then ‘
29:        Add_C(cluster_2, tc)
30:    end if
31: end procedure
```

tasks can be associated to the cluster. The main rules applied at this level are:

- **Rule 1:** $\forall\ \zeta_l, l \in [1..\gamma], \forall\ T_i \in \zeta_l\ i \in [1..\mathcal{R}]$, for each pair of functions $F_{k,i}$ and $F_{h,i}$ / $F_{k,i} \in$ Inclu($F_{h,i}$), group $F_{k,i}$ and $F_{h,i}$ on the same cluster,
- **Rule 2:** $\forall\ \zeta_l, l \in [1..\gamma], \forall\ T_i \in \zeta_l, i \in [1..\mathcal{R}]$, for each pair of functions $F_{k,i}$ and $F_{h,i}$ / $F_{k,i} \in$ Exclu($F_{h,i}$), put $F_{k,i}$ and $F_{h,i}$ on different clusters.

Algorithm 1 describes this partitioning phase where (i) *ttask* is a table containing tasks of a configuration, (ii) *tc* is a table that will hold the constructed clusters, (iii) *tf* is a table that will hold the functions that are not affected with the inclusion/exclusion constraints, (iv) *Cluster_T(c, F, tab)* is a function that stores a function F and all the elements of a table *tab* into a cluster c, (v) *Cluster(c, F)* is a function that stores a function F into a cluster c, (vi) *OK_Memory(c)* is a function that indicates memory availability in a cluster c, (vii) *OK_Energy(c)* is a function that indicates energy availability in a cluster c, (viii) *Add_C(c, tab)* is a function that adds a cluster c to a table *tab*, and (ix) *Add_F(F, ta)* is a function that adds a function F to a table *ta*.

Inclusion/exclusion is a hard constraint, thus clustered elements are locked and they will not be moved to other clusters during the remaining process.

2) HIERARCHICAL PARTITIONING

Clusters the remaining functions that have no inclusion/exclusion constraints. The functions are evaluated by their connecting edges probabilities and high probability values are treated first. For each remained function $F_{k,i}$, all its predecessors are assessed to determine the highest probability value of their connecting edges. $F_{k,i}$ is associated to the cluster where the predecessor having the highest edge probability value is located.

Algorithm 2 Hierarchical Partitioning Algorithm

```
1: procedure Hierarchical-partition(var tc: tab-cluster, tf: tab-function, ttask: tab-task)
2:     for k = 1 to length(tf) do
3:         PT ← FetchTask(ttask, tf[i])
4:         Pred(ttask[PT], tf[k], Tpred)
5:         ok ← false
6:         repeat
7:             max ← maxProba(tf[k], Tpred)
8:             PC ← FetchCluster(tc, Tpred[max])
9:             if OK_Memory(tc[PC]) & OK_Energy (tc[PC]) then
10:                tc[PC] ← tf[k]
11:                ok ← true
12:            else Tpred[max] ← 0
13:            end if
14:        until ok
15:    end for
16: end procedure
```

Algorithm 2 describes this partitioning phase where (i) *ttask* is a table containing tasks of a configuration, (ii) *tc* is a table that will hold the constructed clusters, and (iii) *tf* is a table that will hold the functions that are not affected with the inclusion/execution constraint, (iv) *FetchTask(ttask, F)* is a function that fetches a table of tasks *ttask* in order to determine the index of the task that includes the function F, (v) *maxProba(F, Tpred)* determines the maximum probability value of edges connected to a function F using the table of its predecessors *Tpred*, (vi) *OK_Memory(c, F)* is a boolean function that returns true if there is enough memory on a cluster c for a function F, (vii) *OK_Energy(c)* is a boolean function that returns true if there is enough energy on a cluster c for a function F, (viii) *Pred(ta, func, Tpred)* is a function that returns the predecessors of a function *func* using its corresponding task *ta* and stores them into a table *Tpred*,

and (ix) *FetchCluster(tc, F)* is a function that fetches a table of clusters *tc* in order to determine the index of the cluster storing the function *F*.

The rules to be applied are:

- **Rule 3:** for any remaining un-clustered $F_{k,i} \in T_i$, determines its predecessors $F_{k-1,i}$ in the DAG of T_i,
- **Rule 4:** For any $F_{k,i}$, extracts the highest edge's probability couples $\prec F_{k,i}, F_{k-1,i} \succ$ and cluster $F_{k,i}$ with its related clustered functions having the highest edge probability.

3) Kernighan-Lin

Optimizes the generated clusters. This phase evaluates both probability and communication cost on the edges connecting functions by gain calculation. If the gain is positive, then the function is moved to another cluster.

This step applies the following rules:

- **Rule 5:** starts with choosing an unlocked function $F_{k,i}$,
- **Rule 6:** calculates the gain G_F of moving $F_{k,i}$ from a partition to another, $G_F = ((Cc_e \times Pr_e) - (Cc_h \times Pr_h))$ where (i) Cc_e is the communication cost of edges connecting $F_{k,i}$ with $F_{e,i}$ placed in another cluster, (ii) Cc_h is the communication cost of edges connecting $F_{k,i}$ with $F_{h,i}$ placed in its own cluster, (iii) Pr_e is the probability of edges connecting $F_{k,i}$ with $F_{e,i}$ placed in another cluster, and (iv) Pr_h is the probability of edges connecting $F_{k,i}$ with $F_{h,i}$ placed in its own cluster.
- **Rule 7:** If $G_F \geq 0$ then we move $F_{k,i}$ to another cluster.

Algorithm 3 Kernighan-Lin Optimizationg Algorithm

```
1: procedure Kernighan-Lin Optimization(ttask: tab-task,
   tf: tab-function, tc: tab-cluster)
2:     for h = 1 to length(tf) do
3:         PT ← FetchTask(ttask, tf[h])
4:         Pred(ttask[PT], tf[h], Tpred)
5:         Func ← maxGain(tf[h], Tpred)
6:         if Func ≠ NULL then
7:             PC ← FetchCluster(tc, Func)
8:             if OK_Memory(tc[PC]) & OK_Energy
   (tc[PC]) then
9:                 tc[PC] ← tf[h]
10:                P ← FetchCluster(tc, tf[h])
11:                Remove(tc[P], tf[h])
12:            end if
13:        end if
14:    end for
15: end procedure
```

The kernighan-lin optimization algorithm is described with Algorithm 3 where (i) *ttask* is a table containing tasks of a configuration, (ii) *tc* is a table that will hold the constructed clusters, (iii) *tf* is a table that will hold the functions that are not affected with the inclusion/execution constraints, (iv) *maxGain(F, tab)* is a function that returns a function having a maximum gain value when stored with the function *F* on the same cluster from a table *tab* storing *F* predecessors, and (v) *Remove(c, F)* is a function that removes a function *F* from a cluster *c*.

IV. R-CODESIGN

In this section, we present the R-codesign methodology. A system specification according to the proposed probabilistic modeling DAGs is the input of R-codesign. From these DAGs, hardware and software tasks are extracted and processed through the I-codesign engine. A mapping process follows the I-codesign algorithms and a mapping matrix is generated in order to be used further by the co-simulation module. A validation strategy is then applied. If the performance results from the validation module are not convenient, then the I-codesign module is called again and a new mapping is recalculated. There is no upper bound on the created allocation by the I-codesign module. The designed system should be capable of running different configurations. By applying the proposed methodology, an allocation for each specified configuration is created.

Algorithm 4 R-Codesign Algorithm

```
1: procedure R-codesign(Tconf:tab-configuration,
   length[Tconf]:integer)
2:     if length(Tconf) > 0 then
3:         NbT ← Number(Tconf[length(Tconf)])
4:         for k = 1 to NbT do
5:             TaskExtraction(T_k, DAG_sw, DAG_hw)
6:             repeat
7:                 Mapping_Table ← Icodesign
   (DAG_hw, HW)
8:                 Mapping_Table ← Icodesign
   (DAG_sw, SW)
9:                 PerformanceResults ← CoSimulation
   (Mapping_Table)
10:            until PerformaceResults == "ok"
11:        end for
12:        R − codesgin(Tconf, length[Tconf] − 1)
13:    end if
14:    GenerateController()
15: end procedure
```

Algorithm 4 implements the R-codesign methodology. The input specification can be composed of multiple configurations scenarios where each scenario executes a set of tasks. Thus we define *Tconf* as the configuration table that will be the main input of Algorithm 4 and *Number(conf)* is a function that returns the number of tasks per configuration *conf*. Figure 3 represents the flow diagram of the methodology. R-codesign steps are stated as follows:

A. TASK EXTRACTION

R-codesign starts with extracting software functions and hardware behaviors from the system specification DAGs.

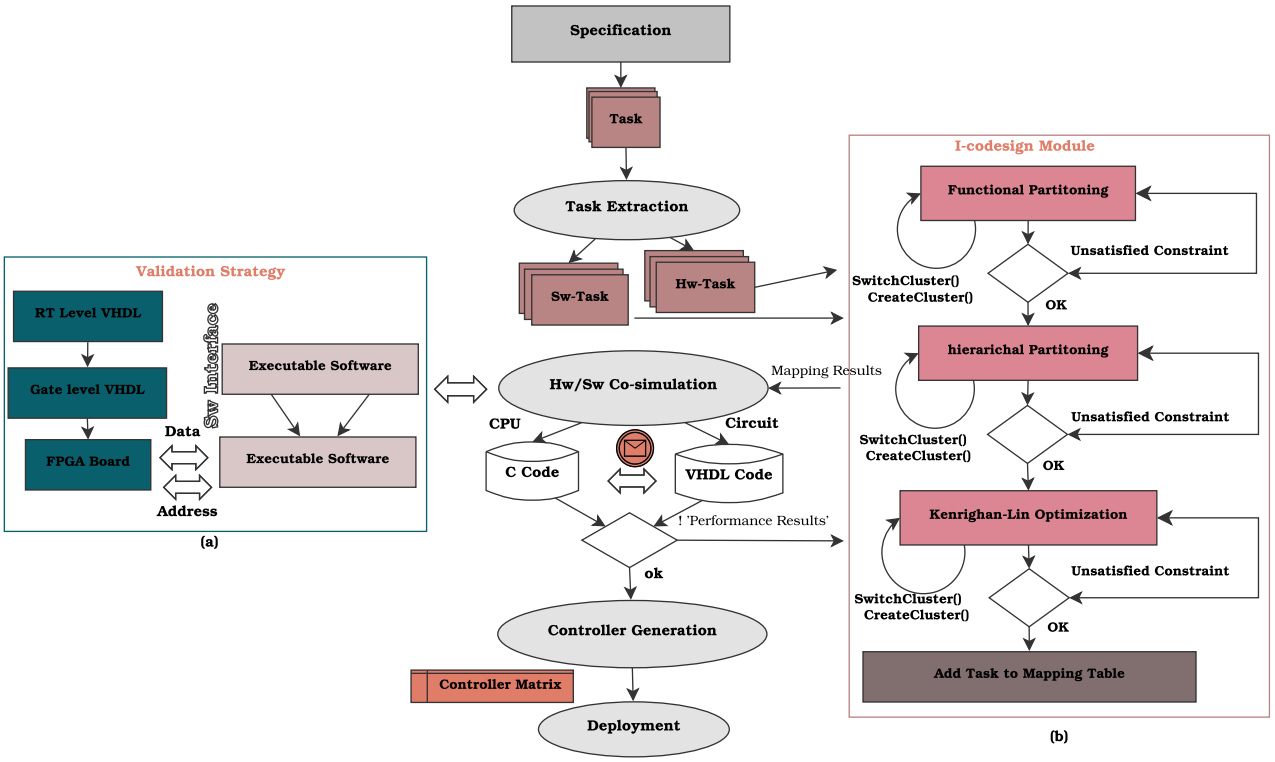


FIGURE 3. R-codesign flow.

It constructs a sub-DAG for each type of task elements. The task extraction is performed as follows.

- $\forall\ \zeta_l,\ l \in [1..\gamma],\ \forall\ T_i \in \zeta_l,\ i \in [1..\mathcal{R}],\ \forall\ F_{k,i}/B_{j,i} \in T_i$, add $\prec F_{k,i}, F_{k-1,i}\succ$ to the software sub-DAG and $\prec B_{j,i}, B_{j-1,i}\succ$ to the hardware sub-DAG.

During the extraction phase, we adjust the communication cost and the probability estimation of behaviors edges that are separated with function(s) or functions edges separated with behavior(s). As for the transfer cost, it is the sum of the individual communication cost of edges separating the communicating functions/behaviors. Regarding the edge probability, it is the product of the individual edge probability separating communicating functions/behaviors. For example, in Figure 2, the probability on the edge connecting $F_{1,i}$ to $F_{2,i}$ is equal to 0.6×0.8 while the communication cost is $10 + 12$. If there are more than one functions/behaviors that communicates with another function/behavior, i.e., more than one incoming edge in the nodes of the graph, then we adjust the communication cost related to the edge that will bind the two behaviors by summing the communication cost on each communicating path and considering the highest value of the communication cost. As to the adjusted probability, it is the sum of probability of the multiple paths (its value is 1 when the sum exceeds 1). As mentioned earlier, the path probability is the product of all the individual edges probability constituting the path.

B. I-CODESIGN FOR HARDWARE BEHAVIORS

The inclusion/exclusion constraints are hard constraints that generally decide the number of clusters to be created and lock the behaviors that are concerned in term of placement. If behaviors share the control of the same components, then they are deployed on the same FPGA. Since each behavior has its own design, FPGAs can have several implementations. Multiplexers can be used in order to switch from an implementation to another. The assignment of the behaviors based on this constraint is formalized as follows.

$$\forall B_{p,i}, B_{j,i} \in Assign(Cl), \quad p, j \in [1..n_{i,1}], \quad B_{p,i} \notin Exclu(B_{j,i}) \quad (1)$$

$$\forall B_{p,i}, B_{j,i}/B_{p,i} \in Inclu(B_{j,i}),\ Then\ B_{p,i}, B_{j,i} \in Assign(Cl) \quad (2)$$

where Cl is a cluster created based on the exclusion/inclusion constraint. The number of the created clusters depends on the number of behaviors on the hardware DAG. $Inclu(B_{j,i})$ designates the behaviors that are related with inclusion to $B_{j,i}$. $Exclu(B_{j,i})$ designates the behaviors that are related with exclusion to $B_{j,i}$. $Assign(Cl)$ groups the set of behaviors affected to the cluster Cl. We define N_{Cl} as the number of elements associated to a cluster Cl.

Each cluster created for hardware tasks $Cl = \{B_{1,i}, B_{2,i}, \ldots, B_{N_{Cl},i}\}$ is composed of behaviors and will be implemented on a single FPGA unit. The reconfigurable hardware

device offers a certain amount of computational resources, e.g., the configurable logic blocks of a FPGA, which is also referred to as the S_{FPGA} parameter of the device. At each iteration of the I-codesign methodology, a placement decision can affect the hardware units. Hence, the available area on the FPGA must be sufficient in order to execute the affected behaviors. Thus, we propose to apply the following constraint:

$$\Sigma_{B_{j,i}\in Cl} M_j^{hw}(B_{j,i}) < S_{FPGA} \tag{3}$$

The third constraint concerns the bandwidth which is correlated to the transmission data rate and expressed in Bytes per seconds. The bandwidth affects the transmission capacity between the linked components (CPUs, FPGAs, IPs) and particularly when there are data dependencies between two tasks located in different hardware units. This dependency constraint is defined as follows: a behavior $B_{p,i}$ placed in a cluster Cl_u depends on a behavior $B_{j,i}$ placed in a cluster Cl_v:

$$\forall B_{p,i} \leftrightarrow B_{j,i},\ B_{p,i} \in Cl_v, B_{j,i} \in Cl_u \tag{4}$$

$$Bandwidth(Cl_v, Cl_u) = \sum_{B_{p,i}\in Cl_v, B_{j,i}\in Cl_u} Bandwidth(B_{p,i}, B_{j,i}) \leq BW_{v,l} \tag{5}$$

where $BW_{v,l}$ stands for an available bandwidth between two tiles Hw_v and Hw_l. $B_{p,i} \leftrightarrow B_{j,i}$ means that $B_{p,i}$ and $B_{j,i}$ are placed on different clusters Cl_u and Cl_v and that they have a data dependency. The expression $Bandwidth(B_{p,i}, B_{j,i})$ corresponds to the bandwidth between $B_{p,i}$ and $B_{j,i}$. The verification step includes also the energy consumed by a given FPGA. Indeed, the energy consumption of a partition Cl depends on the selected operating frequency $Freq$ based on the current configuration and the number of available gates S_{FPGA} of the corresponding tile. The electrical power constraint is given by

$$S_{FPGA}.[Freq]^3 < Pw_{FPGA} \tag{6}$$

In case of unavailable area or energy insufficiency, the problem is reported to the I-codesign methodology and another FPGA is allocated. When all these constraints are satisfied, the real-time feasibility is evaluated.

$$U_k = \sum_{B_{j,i}\in Cl} E_{hw}(B_{j,i})/P_{hw}(B_{j,i}) \leq 1 \tag{7}$$

According to EDF scheduling algorithm, the feasibility is tested using the following Eq. (7) where U_k is the utilization of the cluster Cl. The summary of the constraints that has to be respected at each step of the co-design for the hardware behaviors are given by

$$\begin{cases} \Sigma_{B_{j,i}\in Cl} M_j^{hw}(B_{j,i}) < S_{FPGA} \\ \sum_{B_{p,i}\in Cl_v, B_{j,i}\in Cl_u} Bandwidth(B_{p,i}, B_{j,i}) \leq BW \\ S_{FPGA}.[Freq]^3 < Pw_{FPGA} \\ U_k = \sum_{B_{j,i}\in Cl} E_{hw}(B_{j,i})/P_{hw}(B_{j,i}) \leq 1 \end{cases} \tag{8}$$

C. I-CODESIGN FOR SOFTWARE FUNCTIONS

Previously described equations (1) and (2) are also applicable to the software functions when dealing with the inclusion/exclusion constraints. Each created cluster Cl for software tasks resulting from I-codesign algorithms will be mapped to a CPU unit and is composed of N_{Cl} functions. We assume that all the target MPSoC CPUs are homogeneous. Thus, they have common characteristics (S_{CPU},..). For a given software sub-DAG of T_i composed of $n_{i,2}$ functions, the following constraints should be verified: The available memory on the designed CPUs at an iteration must be sufficient in order to place the affected functions. An extra CPU is designated in case of resource shortage. We define the memory space constraint as follows.

$$\Sigma_{F_{k,i}\in Cl} M_k^{sw}(F_{k,i}) < S_{CPU} \tag{9}$$

The bandwidth constraint is applied by following the equations (4) and (5) using functions $F_{k,i}$ instead of behaviors $B_{j,i}$. The energy constraint is verified using equation (10) where V is the voltage and Ca is the capacitance of the corresponding processor.

$$Freq \times V^2 \times Ca < Pw_{CPU} \tag{10}$$

Real-time feasibility is verified according to the EDF algorithm following the equation below.

$$U_k = \sum_{F_{k,i}\in Cl} E_{sw}(F_{k,i})/P_{sw}(F_{k,i}) \leq 1 \tag{11}$$

where U_k is the utilization of a cluster Cl, E_{sw} is the execution time of $F_{k,i}$ and P_{sw} is the period of $F_{k,i}$. The summary of the constraints that have to be respected at each step of the co-design for the software functions are given by

$$\begin{cases} \Sigma_{F_{k,i}\in Cl} M_k^{sw}(F_{k,i}) < S_{CPU} \\ \sum_{F_{k,i}\in Cl_v, F_{v,i}\in Cl_u} Bandwidth(F_{k,i}, F_{v,i}) \leq BW \\ Freq \times V^2 \times C < Pw_{CPU} \\ U_k = \sum_{F_{k,i}\in Cl} E_{sw}(F_{k,i})/P_{sw}(F_{k,i}) \leq 1 \end{cases} \tag{12}$$

D. HARDWARE SOFTWARE CO-SIMULATION

Hardware/software co-simulation allows to verify the feasibility of mixed hardware/software descriptions in term of timing constraints. Implementing the co-simulation consists in writing a set of HW components in VHDL and a set of SW components (C programs) and linking them together with communication interfaces. Finally, running co-simulation will lead to two important results: (i) the execution status whether it is a success or a failure. (ii) execution trace which can be used for further analysis. The concept of correctness of this verification is defined as follows: The system fails when it reaches an undefined state or its predefined time frame is violated and no time-out action is defined [37]. If the co-simulation fails, then a remapping scheme is calculated using the I-codesign module.

E. CONTROLLER GENERATION

The software specification is divided into configurations. Each configuration has a set of tasks to be executed when initiated. Initially, a boot configuration is loaded. However, a reconfiguration can occur at run-time which requires to reconfigure (replace, re-parameter, change the functionality, etc) of the system tasks. Therefore, a pre-calculated mapping of all the possible configuration scenarios is necessary and will lead to better performance. Thus, we propose to build a controller module that manages the reconfiguration. It acts following internal or external events that induce reconfigurations. When executing the R-codesign process, a matrix is constructed based on the output of each partitioned configuration (task set). In fact, the task mapping to the execution platform is stored in the controller matrix along with its corresponding execution scenario. Hence, when the system is executing and a reconfiguration occurs, the constructed table is consulted and the proper partitioning according to the I-codesign scheme is applied.

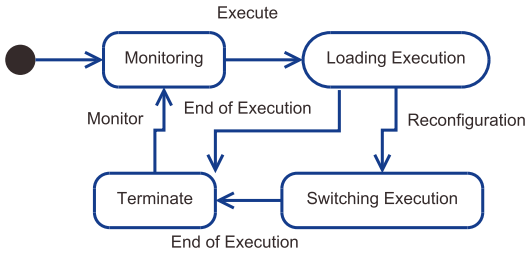


FIGURE 4. The controller State Diagram.

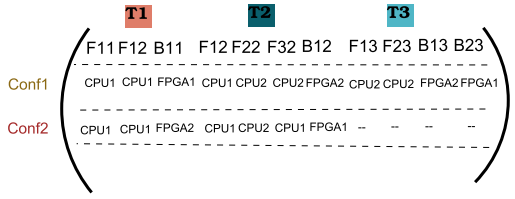


FIGURE 5. The controller Matrix.

The controller receives internal or external events and initiates a necessary reconfiguration. Figure 4 shows the state diagram of the controller. An example of the controller matrix is presented in Figure 5 where the specification is composed of three tasks T_1, T_2 and T_3 and two configurations $conf_1 = \{T_1, T_2, T_3\}$ and $conf_2 = \{T_1, T_2\}$. The number of lines in this matrix is equal to γ the number of possible configurations. The number of columns is equal to $\sum_i^{\mathcal{R}} n_{i,1} + n_{i,2}$. The matrix associates each task function/behavior with a specified PE (Processing Unit: CPU, FPGA) when the corresponding configuration is selected.

V. EXPERIMENTAL RESULTS

A. R-CODESIGN ENVIRONMENT: SPEX

We develop a co-design execution environment that is called SPEX. It provides a toolbox in order to create a hardware/software system description according to the proposed design models and implements the co-design algorithms. It proposes a flexible task set generator for different scenarios and purposes. The tool places the software specification following several design constraints as inclusion/exclusion parameters, probabilistic execution of the software tasks, available memory and energy on the hardware units and real-time parameters. At each iteration, it constructs the controller table that stores all the possible execution scenarios. For simulation purposes the tool loads a specification file, reads the software and hardware characteristics, applies the co-design algorithms and generates the controller table along with memory and energy estimation. Figure 6 summarizes the general tool structure. The tool is composed of four different parts: 1) Task set Generator (TSG), 2) Task Decompositioner (TD), 3) Task Partitioner (TP), 4) Execution Environment (EE). TSG should be set with parameters such as CPU utilization and the desired number of tasks, and then it creates a task set that is called a configuration. The design constraints (probability, inclusion/exclusion, communication costs, and dependency) are randomly generated by the tool. The generated configuration is passed as input to TD which decomposes the tasks into elementary functions with design constraints. TG produces the task graphs. Then, TP performs the partitioning algorithms and generates optimized clusters.

Figure 7 summarizes the TP structure. Finally, EE executes each cluster on the associated computing unit and collects results for reporting energy and memory utilization and controller status. The output results allow the assessment of using static partitioning stored in the controller table (generated by I-codesign) and permits us to compare the I-codesign methodology with a legacy dynamic mapping scheme.

B. A CASE STUDY

We consider a system having two possible configurations denoted by $conf_1$ and $conf_2$. This system is specified by a task graph composed of three tasks T_1, T_2 and T_3 according to R-codesign modeling techniques. The required memory size of the application is 2.3 MB where T_1, T_2 and T_3 require respectively 1.5 MB, 0.5 MB and 0.3 MB. The composition of each configuration is: $conf_1 = \{T_1, T_2\}$ and $conf_2 = \{T_1, T_3\}$. Figure 8 presents the DAG of the task T_1. The target hardware is an MPSoC composed of two identical tiles where each tile has a CPU, a reconfigurable unit and a local memory. The local memory's size is of 1.2 MB.

The reconfigurable device includes one million gates. The master and slave tiles are connected to a shared bus. Arbitration is resolved by a bus arbiter. It periodically examines pending requests from the master and grants access using arbitration mechanisms specified by the bus protocol.

The maximum bandwidth of the communication links is 1 Mbps. Each CPU has a 10 Watt power consumption. The operating frequency *Freq* range is [150..250] Mhz. Software and hardware parameters used in R-codesign phases are listed respectively in Tables 1 and 2. In this case study, the partitioning results as well as the controller generation are of

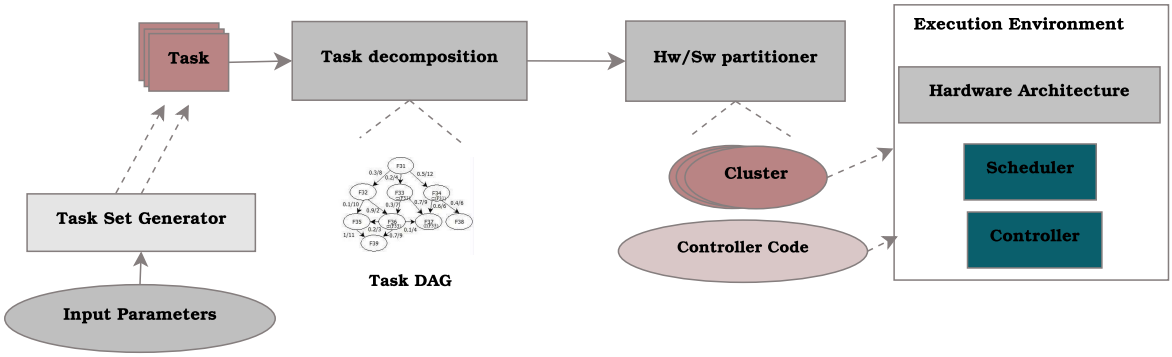


FIGURE 6. The tool Architecture.

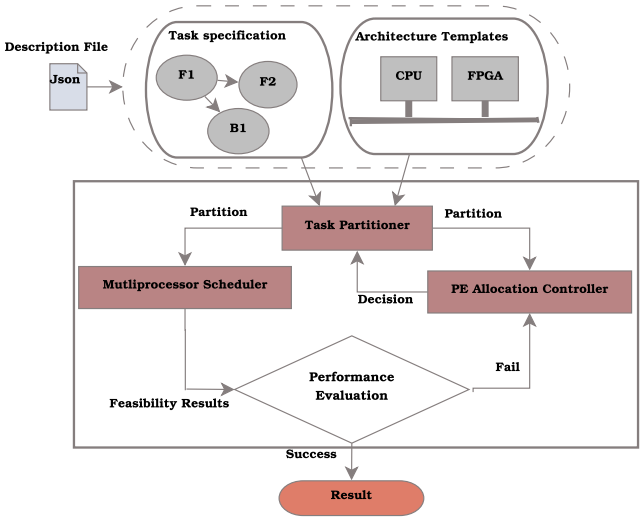


FIGURE 7. The partitioning flow graph of SPEX.

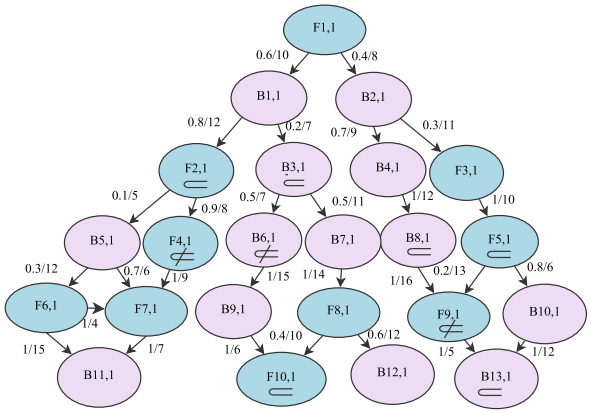


FIGURE 8. The DAG of T_1.

main focus. Following are descriptions of SPEX steps applied to this case study.

1) TASK EXTRACTION

The task extraction builds the software functions and the hardware behaviors graphs.

TABLE 1. Software parameters of T_1.

Function	E_k^{sw}	P_k^{sw}	D_k^{sw}	M_k^{sw}
$F_{1,1}$	5	120	150	0.2
$F_{2,1}$	3	120	200	0.15
$F_{3,1}$	2	90	210	0.11
$F_{4,1}$	6	110	180	0.02
$F_{5,1}$	2	120	190	0.17
$F_{6,1}$	2	200	250	0.05
$F_{7,1}$	2	90	210	0.07
$F_{8,1}$	6	110	180	0.01
$F_{9,1}$	2	120	190	0.08
$F_{10,1}$	2	200	250	0.02

TABLE 2. Hardware parameters of T_1.

Behavior	E_j^{hw}	P_j^{hw}	D_j^{hw}	M_j^{hw}
$B_{1,1}$	3	60	150	0.1
$B_{2,1}$	3	80	200	0.02
$B_{3,1}$	5	90	210	0.05
$B_{4,1}$	5	110	180	0.04
$B_{5,1}$	5	120	190	0.02
$B_{6,1}$	4	160	210	0.035
$B_{7,1}$	1	180	220	0.08
$B_{8,1}$	4	190	260	0.1
$B_{9,1}$	4	160	210	0.02
$B_{10,1}$	1	180	220	0.03
$B_{11,1}$	4	190	260	0.022
$B_{12,1}$	4	190	260	0.01

The probability and the communication cost are recalculated according to the connections roots and leaves on the original task's DAG. Figures 9 and 10 present the extraction of the software and hardware DAGs.

2) FUNCTIONAL PARTITIONING

This step evaluates the inclusion/exclusion constraints and generates initial clusters with locked functions/behaviors.

Then it optimizes the number of generated clusters since their creation depends on these inclusion/exclusion

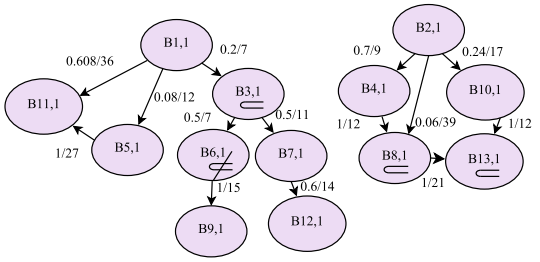


FIGURE 9. Task extraction: Hardware graph.

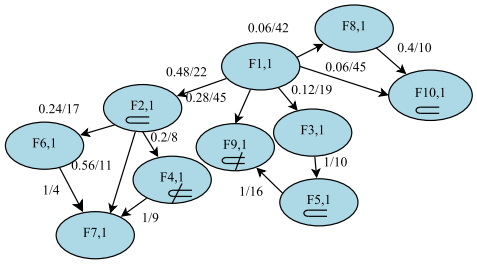


FIGURE 10. Task extraction: Software graph.

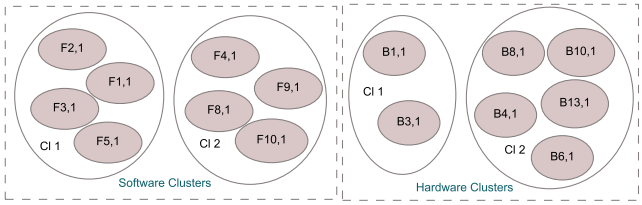


FIGURE 11. Resulted clusters after the Functional Partitioning.

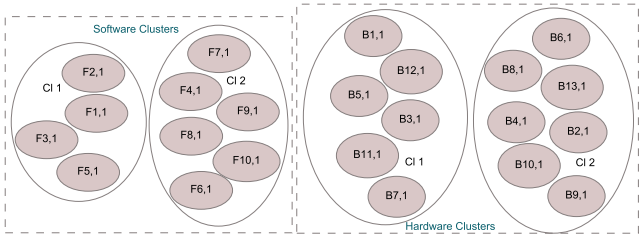


FIGURE 12. Resulted Clusters after the Hierarchical Partitioning.

constraints. The verification phase applied on the created clusters succeeds since the placements meet these constraints (8) and (12). Figure 11 shows the initial clusters created after applying the functional partitioning algorithm on the hardware/software DAGs.

3) HIERARCHICAL PARTITIONING

This phase optimizes the communication costs on communication links since it stores the most probabilistic traffic on the same processor. It also optimizes the processor occupations and assigns tasks to the maximum load of processors. Figure 12 shows the resulted clusters after applying the hierarchical partitioning algorithm.

The verification phase applied on the created clusters succeeds since the placements respect (8) and (12).

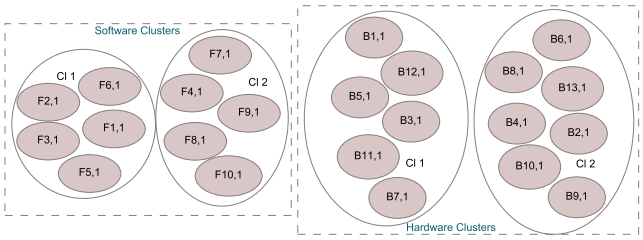


FIGURE 13. Resulted Clusters after the Kernighan Lin Optimization.

4) Kernighan-Lin OPTIMIZATION

This phase aims to optimize the resulting clusters from the hierarchical clustering phase by iterative improvements. In our partitioning process, a combination of two metrics is used in order to optimize the traffic circulation of the system: the communication cost and the probabilistic estimations of the executions. Figure 13 shows the optimized clusters after applying the kernighan-Lin algorithm. The verification phase applied to the optimized clusters succeeds since the placements respect (8) and (12).

5) CONTROLLER GENERATION

For each configuration, SPEX runs R-codesign on the hardware/software specification and constructs the controller matrix. The generated matrix for this case study is showed in Figure 14. The output of SPEX is also presented in Figure 15.

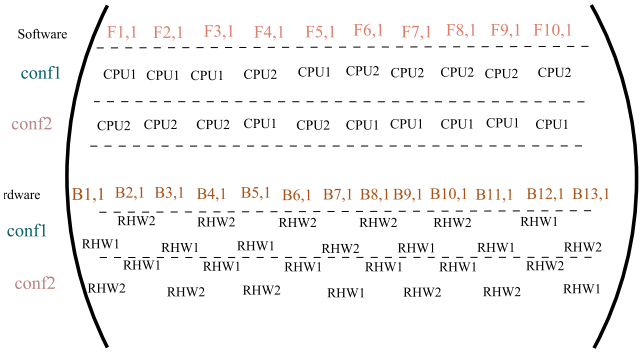


FIGURE 14. Resulted Controller Matrix of task T_1.

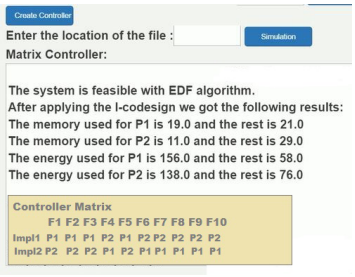


FIGURE 15. Output of SPEX.

C. EVALUATION

To evaluate the R-codesign methodology, several task sets of different dimensions are generated. The generated tasks

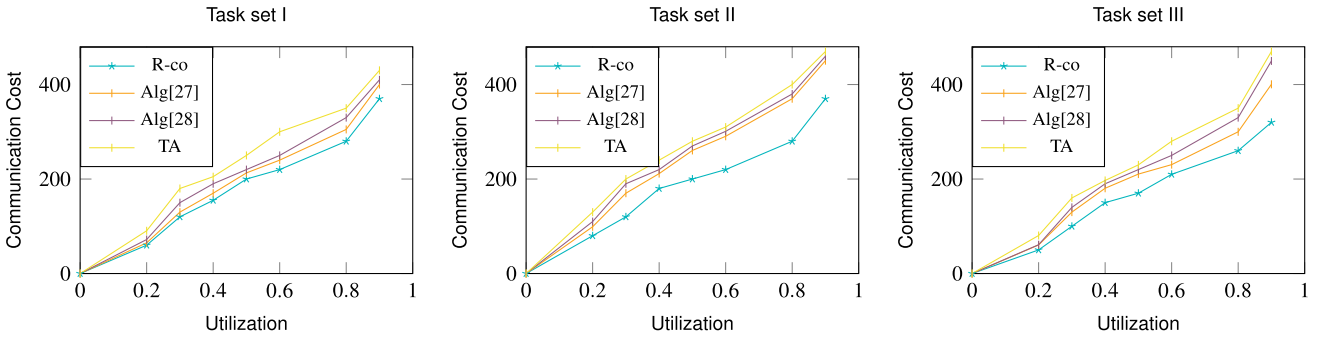


FIGURE 16. Simulation Results for Communication Costs.

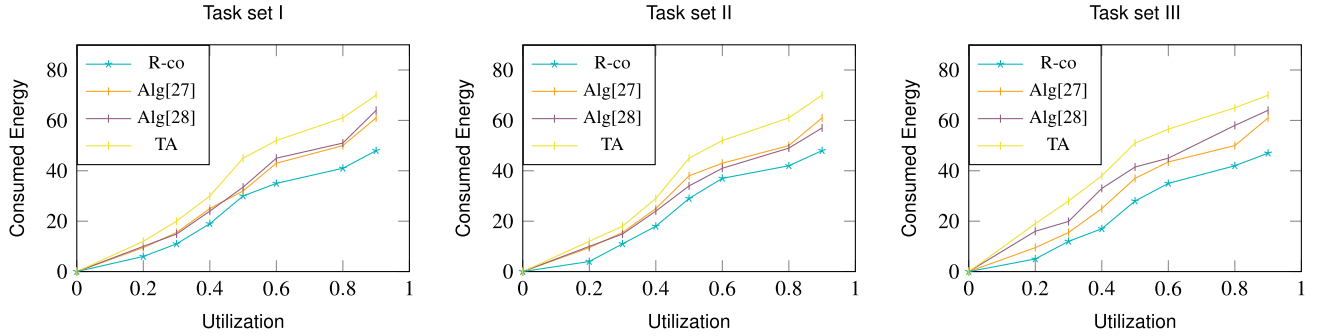


FIGURE 17. Simulation Results for Energy Consumption.

are processed through SPEX and we obtain the generated controller matrix along with the mapping scheme of each execution scenario.

For performance simulation, we use ARTS framework which is a simulation tool for user-driven abstract MPSoC design explorations. Hence, The framework allows to: (i) model processing elements (PE), memory units and interconnect, (ii) investigate PE utilization, memory usage, communication issues, and energy/power consumption, and (iii) analyze the causality between MPSoC components i.e., resource constrains and inter-dependencies [38]. The application model is based on task graphs, where the exact functionality of a task is abstracted away and expressed using a set of timing constraints (execution time, deadline and offset). Recording files are generated providing an overview on the architecture-under-test, the profile of the application, the PE utilization, the memory and communication costs. ARTS model captures the impact of the dynamic and unpredictable behavior on processor, memory and communication performance. In particular, it focuses on analyzing the impact of application mapping on the processor and memory utilization taking the on-chip communication latency into account.

The evaluated performance parameters that are taken into account are the total communication costs of the system functions/behaviors, the total consumed energy during the system execution, the total number of exchanged messages, and the total execution time of the grouped task sets. The generated results when varying the utilization on CPUs and FPGAs are compared with two partitioning and scheduling algorithms: the work reported in [27] which proposes a task allocation algorithm based on clustering. This work that finds a near optimal solution and tries to minimize the total system cost by forming a cluster of tasks in such a way that the cluster, having minimum execution cost, is allocated first. Then comes the second work [28] which proposes an algorithm to extend the battery life by partitioning and scheduling the input task wisely. We also compare the performance results from the traditional approach *TA* that during the run-time reconfiguration calculates the appropriate mapping of tasks into processors with R-codesign *R-co*.

Figures 16, 17, 18 and 19 present the performance resulting from randomly generated task sets. Figure 16 describes the communication costs in term of delays of the transfer through the communication medium while Figure 19 enumerates the transferred messages. The comparison between the evaluated approaches has demonstrated that R-codesign offers better performance results particularly with large utilization factors and high number of nodes on the specification DAGs. These enhancements are due to probabilistic estimation of the communicated functions/behaviors that store dependent tasks with high chances to be executed successively on same PEs. Another advantage of R-codesign is the pre-calculated mapping of the possible reconfigurations at run-time. This step helps significantly to minimize the reconfiguration overhead which is made clear from the comparison with *TA*. Due to these changes a reduction by 30% of the

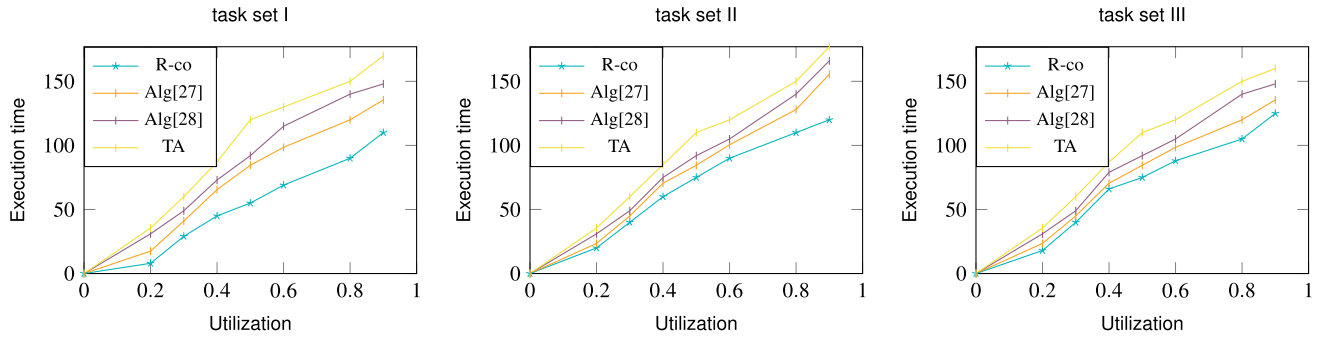


FIGURE 18. Simulation Results for Execution Time.

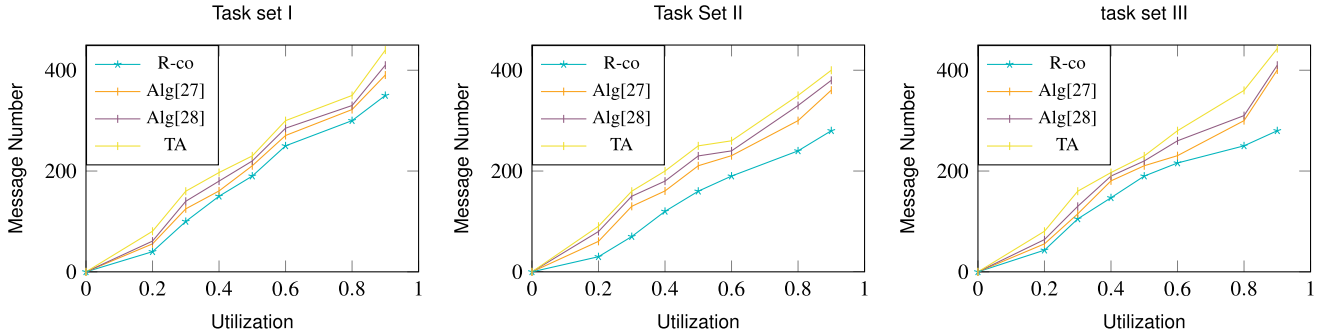


FIGURE 19. Simulation Results for the number of exchanged messages.

global execution time is observed. Since execution time is a crucial performance parameter of the embedded system design, adopting the proposed idea improves significantly the response time. Compared with existing works, it is shown from the graphs that R-codesign improves the communication costs with an average of 10% and therefore the exchanged messages are reduced by an average of 12%. Simulation results show that this contribution has few benefits: (i) the number of exchanged messages has been noticeably reduced, and (ii) the global execution time has been minimized. Hence, the energy consumption will be reduced as a result of decreased execution time. Another advantage of R-codesign is its validation tests (see equation systems (8) and (12)) that avoid any issues related to a lack of resources.

VI. CONCLUSION

In this paper, we propose a complete methodology for modeling, partitioning and validating reconfigurable embedded system design. We expose in this paper probabilistic estimation of the executions and a mathematical formalization of the design constraints. The obtained performance improvement of the proposed techniques in terms of communication costs (the number of exchanged messages), consumed energy and required CPU time has been verified. Furthermore, the new partitioning combination of iterative, constructive and functional techniques allows efficient and optimized placements of software/hardware specification while respecting the constrained resources. We proposed an execution model for R-codesign methodology that relies on a controller module that stores all the possible reconfiguration scenarios and manages the system tasks whenever a reconfiguration event occurs. Finally, we developed the SPEX tool which allows to: (i) write a specification according to the R-codesign system model, (ii) apply the new partitioning techniques and (iii) generate the controller table. We are working on enhancing SPEX and from a future perspective we intend to explore tasks migration mechanisms for an optimal reconfiguration process [39]. Future work includes considering the exsitence of faults [40] in a reconfigurable embedded system and the case of partially known system models [41] using a Petri net models or finite state automata.

REFERENCES

[1] M. Uzam, Z. Li, G. Gelen, and R. S. Zakariyya, "A divide-and-conquer-method for the synthesis of liveness enforcing supervisors for flexible manufacturing systems," *J. Intell. Manuf.*, vol. 27, no. 5, pp. 1111–1129, Oct. 2016.

[2] Y. Chen, Z. Li, K. Barkaoui, and M. Uzam, "New Petri net structure and its application to optimal supervisory control: Interval inhibitor arcs," *IEEE Trans. Syst., Man, Cybern., Syst.*, vol. 44, no. 10, pp. 1384–1400, Oct. 2014.

[3] T. Kim and S. Tak, "Experience with hardware-software codesign of network protocol stacks supporting real-time inter-task communication," in *Proc. IEEE 10th Int. Conf. Comput. Inf. Technol. (CIT)*, Bradford, U.K., Jun. 2010, pp. 26–32.

[4] S. Zhang, N. Wu, Z. Li, T. Qu, and C. Li, "Petri net-based approach to short-term scheduling of crude oil operations with less tank requirement," *Inf. Sci.*, vol. 417, pp. 247–261, Nov. 2017.

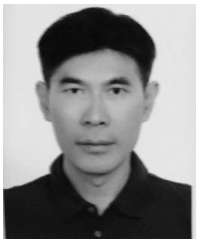
[5] H. Grichi, O. Mosbahi, M. Khalgui, and Z. Li, "RWiN: New methodology for the development of reconfigurable WSN," *IEEE Trans. Autom. Sci. Eng.*, vol. 14, no. 1, pp. 109–125, Jan. 2017.

[6] M. Gasmi, O. Mosbahi, M. Khalgui, L. Gomes, and Z. Li, "R-node: New pipelined approach for an effective reconfigurable wireless sensor node," *IEEE Trans. Syst., Man, Cybern., Syst.*, to be published.
[7] H. Grichi, O. Mosbahi, M. Khalgui, and Z. Li, "New power-oriented methodology for dynamic resizing and mobility of reconfigurable wireless sensor networks," *IEEE Trans. Syst., Man, Cybern., Syst.*, to be published.
[8] M. O. Ben Salem, O. Mosbahi, M. Khalgui, Z. Jlalia, G. Frey, and M. Smida, "BROMETH: Methodology to design safe reconfigurable medical robotic systems," *Int. J. Med. Robot. Comput. Assist. Surg.*, vol. 13, no. 3, p. e1786, 2017.
[9] F. Ferrandi, P. L. Lanzi, C. Pilato, D. Sciuto, and A. Tumeo, "Ant colony optimization for mapping, scheduling and placing in reconfigurable systems," in *Proc. NASA/ESA Conf. Adapt. Hardw. Syst. (AHS)*, Turin, Italy, Jun. 2013, pp. 47–54.
[10] X. Wang, I. Khemaissia, M. Khalgui, Z. Li, O. Mosbahi, and M. Zhou, "Dynamic low-power reconfiguration of real-time systems with periodic and probabilistic tasks," *IEEE Trans. Autom. Sci. Eng.*, vol. 12, no. 1, pp. 258–271, Jan. 2015.
[11] T.-K. Liu, Y.-P. Chen, and J.-H. Chou, "Developing a multiobjective optimization scheduling system for a screw manufacturer: A refined genetic algorithm approach," *IEEE Access*, vol. 2, pp. 356–364, 2014.
[12] W. Housseyni, O. Mosbahi, M. Khalgui, Z. Li, and L. Yin, "Multiagent architecture for distributed adaptive scheduling of reconfigurable real-time tasks with energy harvesting constraints," *IEEE Access*, to be published.
[13] S. Brandstätter and M. Huemer, "A novel MPSoC interface and control architecture for multistandard RF transceivers," *IEEE Access*, vol. 2, pp. 771–787, 2014.
[14] S. Dutt, "New faster Kernighan-Lin-type graph-partitioning algorithms," in *Proc. Int. Conf. Comput. Aided Design (ICCAD)*, Nov. 1993, pp. 370–377.
[15] D. W. Franke and M. K. Purvis, "Design automation technology for codesign: Status and directions," in *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, May 1992, pp. 2669–2672.
[16] K. Li, X. Tang, B. Veeravalli, and K. Li, "Scheduling precedence constrained stochastic tasks on heterogeneous cluster systems," *IEEE Trans. Comput.*, vol. 64, no. 1, pp. 191–204, Jan. 2015.
[17] C.-C. Kao, "Performance-oriented partitioning for task scheduling of parallel reconfigurable architectures," *IEEE Trans. Parallel Distrib. Syst.*, vol. 26, no. 3, pp. 858–867, Mar. 2015.
[18] R. Niemann and P. Marwedel, "An algorithm for hardware/software partitioning using mixed integer linear programming," *Des. Autom. Embedded Syst.*, vol. 2, no. 2, pp. 165–193, Mar. 1997.
[19] J. Grode, P. V. Knudsen, and J. Madsen, "Hardware resource allocation for hardware/software partitioning in the LYCOS system," in *Proc. Design, Autom. Test Europe*, Feb. 1998, pp. 22–27.
[20] K. S. Chatha and R. Vemurl, "MAGELLAN: Multiway hardware-software partitioning and scheduling for latency minimization of hierarchical control-dataflow task graphs," in *Proc. 9th Int. Symp. Hardw./Softw. Codesign (CODES)*, Apr. 2001, pp. 42–47.
[21] N. N. Binh, M. Imai, A. Shiomi, and N. Hikichi, "A hardware/software partitioning algorithm for designing pipelined ASIPs with least gate counts," in *Proc. 33rd Design Autom. Conf.*, Jun. 1996, pp. 527–532.
[22] G. Quan, X. Hu, and G. Greenwood, "Preference-driven hierarchical hardware/software partitioning," in *Proc. IEEE Int. Conf. Comput. Design, VLSI Comput. Process*, Oct. 1999, pp. 652–657.
[23] P. Arató, S. Juhász, Z. Á. Mann, A. Orbán, and D. Papp, "Hardware-software partitioning in embedded system design," in *Proc. IEEE Int. Symp. Intell. Signal Process.*, Sep. 2003, pp. 197–202.
[24] C. Wang, A. Wang, Y. Chen, A. Al-Ahmari, and Z. Li, "On computation reduction of liveness-enforcing supervisors," *IEEE Access*, vol. 5, pp. 14775–14786, Aug. 2017.
[25] C. Li, Y. Chen, Z. Li, and K. Barkaoui, "Synthesis of liveness-enforcing Petri net supervisors based on a think-globally-act-locally approach and vector covering for flexible manufacturing systems," *IEEE Access*, vol. 5, pp. 16349–16358, Jun. 2017.
[26] J. Shi, W. Liu, M. Jiang, H. Che, and L. Chen, "Software hardware co-simulation and co-verification in safety critical system design," in *Proc. IEEE Int. Conf. Intell. Rail Transp. (ICIRT)*, Beijing, China, Aug./Sep. 2013, pp. 71–74.
[27] P. Bhardwaj and V. Kumar, "An effective load balancing task allocation algorithm using task clustering," *Int. J. Comput. Appl.*, vol. 77, no. 7, pp. 32–39, Sep. 2013.
[28] R. Shi, S. Yin, C. Yin, L. Liu, and S. Wei, "Energy-aware task partitioning and scheduling algorithm for reconfigurable processor," in *Proc. IEEE 11th Int. Conf. Solid-State Integr. Circuit Technol. (ICSICT)*, Beijing, China, Oct. 2012, pp. 1–3.
[29] X. Wang, Z. Li, and W. M. Wonham, "Dynamic multiple-period reconfiguration of real-time scheduling based on timed des supervisory control," *IEEE Trans. Ind. Informat.*, vol. 12, no. 1, pp. 101–111, Feb. 2016.
[30] G. Yi, J. H. Park, and S. Choi, "Energy-efficient distributed topology control algorithm for low-power IoT communication networks," *IEEE Access*, vol. 4, pp. 9193–9203, 2016.
[31] G. Estrin, R. S. Fenchel, R. R. Razouk, and M. K. Vernon, "SARA (System ARchitects Apprentice): Modeling, analysis, and simulation support for design of concurrent systems," *IEEE Trans. Softw. Eng.*, vol. SE-12, no. 2, pp. 293–311, Feb. 1986.
[32] C. U. Smith, G. A. Frank, and J. L. Cuardrado, "An architecture design and assessment system for software/hardware codesign," in *Proc. 22nd ACM/IEEE Design Autom. Conf.*, Piscataway, NJ, USA, Jun. 1985, pp. 417–424.
[33] A. Kalavade and E. A. Lee, "A hardware-software codesign methodology for DSP applications," *IEEE Des. Test Comput.*, vol. 10, no. 3, pp. 16–28, Sep. 1993.
[34] F. Rose, T. Carpenter, S. Kumar, J. Shackleton, and T. Honeywell, "A model for the coanalysis of hardware and software architectures," in *Proc. 4th Int. Workshop Hardw./Softw. Co-Design*, Washington, DC, USA, Mar. 1996, pp. 94–103.
[35] V. Boppana, S. Ahmad, I. Ganusov, V. Kathail, V. Rajagopalan, and R. Wittig, "Ultrascale+ Mpsoc and FPGA families," in *Proc. IEEE Hot Chips Symp. (HCS)*, Cupertino, CA, USA, Aug. 2015, pp. 1–37.
[36] I. Ghribi, R. B. Abdallah, M. Khalgui, and M. Platzner, "New co-design methodology for real-time embedded systems," in *Proc. 11th Int. Conf. Softw. Eng. Appl.*, 2016, pp. 185–195.
[37] R. Gumzej and M. Colnaric, "An approach to modeling and verification of real-time systems," in *Proc. 4th IEEE Int. Symp. Object-Oriented Real-Time Distrib. Comput.*, 2001, pp. 283–290.
[38] P. Chandraiah and R. Doemer, "Designer-controlled generation of parallel and flexible heterogeneous MPSoC specification," in *Proc. 44th ACM/IEEE Design Autom. Conf.*, San Diego, CA, USA, Jun. 2007, pp. 787–790.
[39] A. R. Gaiduk and N. N. Prokopenko, "Design of nonlinear optimal systems on basis of controlled Jordan form," in *Proc. IEEE East-West Design Test Symp. (EWDTS)*, Sep./Oct. 2017, pp. 1–4.
[40] X. Y. Cong, M. P. Fanti, A. M. Mangini, and Z. W. Li, "Decentralized diagnosis by Petri nets and integer linear programming," *IEEE Trans. Syst., Man, Cybern., Syst.*, to be published, doi: 10.1109/TSMC.2017.2726108.
[41] H. M. Zhang, L. Feng, N. Q. Wu, and Z. W. Li, "Integration of learning-based testing and supervisory control for requirements conformance of black-box reactive systems," *IEEE Trans. Autom. Sci. Eng.*, vol. 15, no. 1, pp. 2–15, Jan. 2018.

INES GHRIBI was born in Testour, Tunisia, in 1990. She received the Engineering degree in computer network and telecommunication from the National Institute of Applied Science and Technology, where she is currently pursuing the Ph.D. degree with the Computer Laboratory of Industrial Systems. Her main research interest is in the design of reconfigurable real-time embedded systems.

RIADH BEN ABDALLAH received the M.Sc. and Ph.D. degrees in computer science from INSA of Lyon, Lyon, France, in 2007 and 2010, respectively. He is currently an Assistant Professor of computer science with the University of Carthage, Tunisia. His current research interests include high performance computing, parallel programming, and embedded software design.

MOHAMED KHALGUI received the B.S. degree in computer science from Tunis El Manar University, Tunis, Tunisia, in 2001, the M.S. degree in telecommunication and services from Henri Poincaré University, Nancy, France, in 2003, the Ph.D. degree from the National Polytechnic Institute of Lorraine, Nancy, France, in 2007, and the Habilitation Diploma degree in information technology (computer science) from the Martin Luther University of Halle-Wittenberg, Halle, Germany, in 2012, with a Humboldt grant. He was a Researcher of computer science with the Institut National de Recherche en Informatique et Automatique INRIA, France; ITIA-CNR Institute, Vigevano, Italy; Systems Control Laboratory, Xidian University, Xi'an, China; and the King Abdulaziz City for Science and Technology, Riyadh, Saudi Arabia; a collaborator with the SEG Research Group, Patras University, Patras, Greece; the Director of RECS Project, O3NEIDA, Canada; the Director of RES Project, Synesis Consortium, Lomazzo, Italy; the Manager of Cyna-RCS Project with Cynapsys Consortium, France; and the Director of BROS and RWiN Projects with ARDIA Corporation, Germany. He is currently a Professor with the University of Carthage, Tunis. He has been involved in various international projects and collaborations. He is a member of various TPC of conferences and many journal boards.



ZHIWU LI (M'06–SM'07–F'16) received the B.S. degree in mechanical engineering, the M.S. degree in automatic control, and the Ph.D. degree in manufacturing engineering from Xidian University, Xi'an, China, in 1989, 1992, and 1995, respectively. He joined Xidian University in 1992. He is currently with the Institute of Systems Engineering, Macau University of Science and Technology, Taipa, Macau. His current research interests include Petri net theory and application, supervisory control of discrete event systems, workflow modeling and analysis, system reconfiguration, game theory, and data and process mining. He is listed in *Marquis Who's Who in the World* (27th Edition, 2010). He is the Founding Chair of the Xi'an Chapter of the IEEE Systems, Man, and Cybernetics Society.

KHALID ALNOWIBET received the Ph.D. degree in operations research from the College of Engineering, North Carolina State University, NC, USA, in 2004. Since 2006, he has been with the Department of Statistics and Operations Research, King Saud University. His interests are queueing theory and its applications, queuing networks, communication networks, and networked embedded system design.

MARCO PLATZNER received the Diploma and Ph.D. degrees in telematics from Graz University of Technology, Graz, Austria, in 1991 and 1996, and the Habilitation degree in hardware-software codesign from Eidgenïossische Technische Hochschule Zurich, Zurich, Switzerland, in 2002. He is currently a Professor of computer engineering with the University of Paderborn, Paderborn, Germany. He is a Faculty Member with the International Graduate School Dynamic Intelligent Systems, University of Paderborn, and the Advanced Learning and Research Institute, Universita della Svizzera Italiana, Lugano, Switzerland. His current research interests include reconfigurable computing, hardware-software codesign, and parallel architectures. He was a Board Member of the Advanced System Engineering Center, University of Paderborn. He is a member of the ACM and serves on the program committees of several international conferences, including FPL, FPT, RAW, ERSA, and DATE. He is a Board Member of the Paderborn Center for Parallel Computing. He is an Associate Editor of the *International Journal of Reconfigurable Computing*, the *EURASIP Journal on Embedded Systems*, and the *Journal of Electrical and Computer Engineering*.