Towards System Level Runtime Design Space Exploration of Reconfigurable Architectures

Kamana Sigdel, Mark Thompson[1], Andy D. Pimentel[1], Koen Bertels

Computer Engineering, EEMCS
Delft University of Technology, The Netherlands
{kamana,koen}@ce.et.tudelft.nl

[1]Computer Architecture Systems Group
University of Amsterdam, The Netherlands
{mthompsn,andy}@science.uva.nl

Abstract—

The ever increasing intricacy of the systems and the increasing use of reconfigurble heterogeneous devices significantly enlarges the design complexity of the modern embedded systems. As a result, to create a good design, it is essential to perform Design Space Exploration(DSE) at various design levels in order to evaluate several design choices. DSE at early design stages helps designers to systematically explore trade-offs between various design goals and to make various design decisions such as hardware/software partitioning, architecture – to – application mappings, task scheduling and task allocation, performance evaluation etc. As the design progresses, the design space can be gradually trimmed and pruned at different design levels of unsuitable design alternatives until a final optimal solution is reached. In this work, we present a system level framework for higher level runtime design space exploration of reconfigurable architectures.

Keywords: **Reconfigurable Computing, Design Space Exploration, System Level Modeling**

I. Introduction and Background

Reconfigurable computing (RC) is becoming increasingly popular as it bears the promise of combining the flexibility of software with the performance of hardware. Reconfigurable devices can serve as runtime re-usable devices for performance critical systems, which allows the reduction of the hardware resources required [1]. Generally, the reconfigurable system consists of traditional microprocessor and a reconfigurable hardware. Hardware implementation of a function in general can have better performance than software implementation, thus, moving selected software region to the reconfigurable hardware can improve the performance of the whole system. Therefore, the reconfigurable systems benefit by speeding up the whole application through implementation of selected application kernels onto reconfigurable hardware [2].

The runtime reconfiguration allows the dynamic or runtime configuration of the hardware, which provides the ability to change the hardware as they are needed during the program execution. Dynamic reconfiguration has the flexibility to accelerate large portion of the application on the same hardware, however, it also introduces reconfiguration latencies. With partial reconfiguration, several tasks can be configured on the hardware individually without interfering with the other tasks running on the same hardware at different stages. As partial reconfiguration configures only a part of the hardware, the computation of one task can be overlapped with the configuration of other task, as a result the configuration latency can be significantly reduced.

The emergence of partial dynamic reconfiguration has added new dimensions to the reconfigurable computing, however, at the same time, it has added new set of challenges to the designers in evaluating the performance of such systems. As a result, the traditional exploration and mapping methods, which only assign the tasks to the fixed hardware and software are not sufficient for designing reconfigurable systems, as they cannot address how the reconfigurable space can be efficiently used to accelerate the tasks.

While designing the dynamic reconfigurable systems, there are several issues that need to be addressed, which are listed as follows:

- **Hardware Software Partitioning**: It deals with the identifying and assigning application or part of ap-

plication onto heterogeneous set of architectural units such as ASICs, General purpose processor (GPP), reconfigurable processor(RP) and so forth. *Spatial partitioning* is the process of identifying the part of the application which can be implemented onto the reconfigurable hardware (HW tasks) and which should be executed as software (SW tasks). The reconfigurable nature of the reconfigurable devices allows it to map the application that is larger than the physical size of the hardware. As a result, for the implementation of applications which requirement exceeds the capacity of the hardware, it is necessary to perform temporal partitioning. *Temporal partitioning* divides the design into mutually exclusive, limited size segments such that requirement for implementing each segment is less than or equal to the capacity of the hardware. It partitions the HW tasks into mutually exclusive "configurations" that will be sequentially executed on the reconfigurable devices at each configuration time.

• **Design Space Exploration**: Design Space Exploration(DSE) allows to systematically explore tradeoffs between various design goals and find the optimal solution. While designing reconfigurable systems, in order to explore all the possible design choices and select an appropriate design point it is essential to perform DSE at various design level. The design point should be determined based on the various system constraints imposed on the system. One example of such design point is the best execution time and area trade-off.

• **Task Allocation**: Task allocation deals with allocating a hardware task onto a reconfigurable logic resources. The efficiency of the reconfigurable systems is also effected by how tasks are allocated and placed on the hardware for execution. By changing the order of task allocation and/or altering the task placement on the hardware can change the reconfiguration overhead.

• **Task Scheduling**: Task scheduling is another problem which deals with scheduling of the various tasks on the architecture. The software tasks, the hardware tasks and the communication channels (shared by the general purpose processor, reconfigurable hardware and other architectural components) have to be scheduled dynamically in order to meet the execution criteria and to avoid the various resource conflicts.

The expanding sophisticated user functionalities and at the same time the increasing use of reconfigurable heterogeneous platforms, significantly enlarges the design space of the modern system. In order to construct a good design and to identify the optimal design choices, it is necessary to explore many design alternatives. However, when using traditional design methods and tools, it is difficult to estimate, analyze or evaluate the performance impact of systems including such reconfigurable logic devices into a system design. Thus, recent demand while designg such system is to have a comprehensive methods which can guide the designers at early design stages for rapid exploration of design in order to make various prudent decisions such as hw/sw partitioning, architecture – to – application mappings, task scheduling and task placing, performance evaluation etc.

In this work, we present a system-level framework for higher level runtime design space exploration, which can assist designers at very early stages in design to perform rapid exploration of different reconfigurable design alternatives. Furthermore, we extend the framework to address the dynamic and adaptive systems where applications as well as architecture can evolve over time. In such dynamic cases, the design process becomes more sophisticated as all the design decision has to be carried out dynamically and the system has to be optimized in terms of runtime behaviors. The framework integrates various techniques related to reconfigurable systems design such as dynamic partitioning, runtime DSE, runtime mapping, dynamic scheduling and placement. For this the Sesame framework has been extended to support partially dynamic reconfigurable architecture of Molen. This work has been presented in our previous paper [3].

The organization of the paper is the following: Section II presents the context for this work. Section III and IV presents the description of the Molen architecture and Sesame simulation framework respectively. Section V discusses our system framework. And, finally, section VI presents the summary and the future work.

II. System Context

This work has been carried out in the context of Delft Workbench [4]. Delft Workbench is a platform for hardware-software co-design assisting designers in tackling various challenges while designing recon-

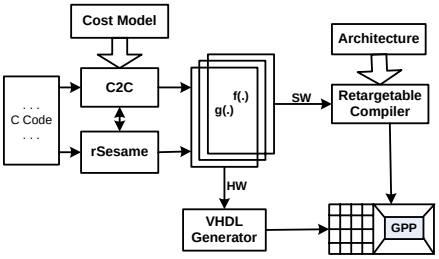

Fig. 1
THE DELFTWORKBENCH PLATFORM FOR HARDWARE SOFTWARE CO-DESIGN

figurable heterogeneous systems. It addresses optimal and rapid designing of reconfigurable embedded systems starting from high-level descriptions and involves various phases such as program analysis, hardware cost estimation, design space exploration, hardware software partitioning, compilation for reconfigurable platform and VHDL generation. The Delftworkbench platform is shown in Figure 1. The scope of this work lies in the module *rSesame* as shown in the figure.

One of the major requirements for the project such as Delft Workbench while designing heterogeneous reconfigurable systems is to obtain the optimal system with efficient utilization of the reconfigurable logic resources. With the increasing design space many choices has to be evaluated and judged before making any kind of design decisions at every design stage. Performing Design Space Exploration at various design levels helps to explore trade-offs between various design goals and to find optimal solutions. As design progresses further, the design space is gradually pruned at different levels in order to find a final optimal solution.

Our focus of the design space exploration is at very higher level where the design to be explored is enormous as no design decisions have yet been made. Making design choices at higher level can rapidly prune the design spaces. In the system level design space exploration, designers can investigate and explore system at early design stages and evaluate the performance very early in the design process. At this level, system behavior (application behavior or architecture characteristics) is represented using several abstraction models. These models are relatively easier and faster to construct, as a result, designer can apply this to traverse a large design. These system-level implementations are evaluated and compared one after another at high level of abstraction and a set of candidates are identified. These candidate sets can further be analyzed at lower abstraction level (such as synthesizable register level (RTL)) in order to reach the optimal solution. Thus, performing DSE at higher level of abstraction facilitates design decisions to be made at very early stages, which can significantly reduce overall design time.

III. THE RECONFIGURABLE ARCHITECTURE

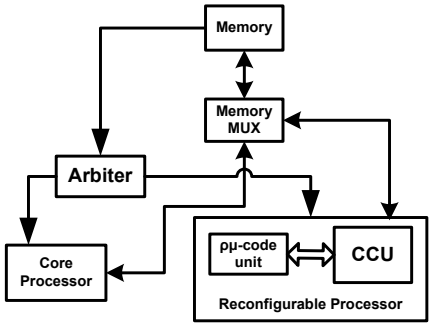

Fig. 2
THE MOLEN ARCHITECTURE

Though our exploration framework is not restricted to a particular type of reconfigurable architecture, for the evaluation purpose, in this research, we use Molen as an example of reconfigurable architecture. The Molen polymorphic processor is established on the basis of the tightly coupled co-processor architectural paradigm [5][6]. The two main components in the Molen machine organization are the 'Core Processor', which is a General Purpose Processor(GPP) and the 'Reconfigurable Processor' (RP). The reconfigurable processor is further subdivided into the $\rho\mu$-code unit and *custom configured unit* (CCU). The CCU consists of reconfigurable hardware, e.g., a field-programmable gate array (FPGA), and memory. GPP and RP are connected to one 'Arbiter' which controls the co-ordination of the GPP and RP (see fig 2). Instructions are issued to either of these processors by the arbiter. In order to speed up the program by running on the reconfigurable hardware, parts of the application running on a GPP can be implemented on the CCU. The code to be mapped onto the reconfigurable hardware is annotated with special pragma directives. When arbiter receives the *pragma* instruction for the

hardware execution, it initiates the reconfigurable operation signal to the reconfigurable unit, gives the data memory control to the RP and drives GPP into a wait state. When arbiter receives an end of reconfigurable signal, it releases back the data memory control back to the GPP and GPP can resume its execution. An operation executed by the RP, is divided into two distinct phases: *set* and *execute*. In the *set* phase, the CCU is configured to perform the supported operations and in *execute* phase the actual execution of the operation is performed.

IV. Sesame Framework

We use Sesame framework as a modeling and simulation platform. Sesame is a DSE environment which facilitates designers at system level exploration of the complex embedded multimedia architectures. In this context, the Sesame environment has been extended in order to model the dynamic reconfigurable behavior of the reconfigurable architectures [3]. The Sesame modeling and simulation environment [7][8] is geared towards fast and efficient exploration of embedded multimedia architectures, typically those implemented as heterogeneous MPSoCs. Sesame adheres to a transparent simulation methodology where the concerns of application and architecture modeling are separated. An application model describes the functional behavior of an application and an architecture model defines the architectural resources and constraints.

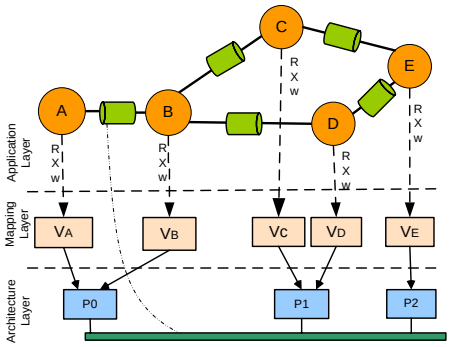

Fig. 3
Three Layers in Sesame Framework

For application modeling, Sesame uses the Kahn Process Network (KPN) model of Computation [9] which is suitable for modeling stream-based (multimedia) applications. This application model consists of concurrent Kahn processes that communicate using blocking read/non-blocking write synchronization over unbounded Kahn channels. The application processes models contain functional application code together with annotations. The application model generates event traces : Read (R), Write (W) and Execute (EX). The events Read (R) and Write(W) are communication event and they describe communication of the Kahn channel between two Kahn processes. And, the Execute (EX) event is a computation event and it describes the computation performed by the Kahn process (typically a function). These events generated by each process while executing the application, are collected into event traces. These traces are mapped onto an architecture model using an intermediate layer called - mapping layer. The Figure 3 shows this mapping with Sesame's three layers: the Application Layer, the Mapping Layer and the Architecture Layer.

The intermediate mapping layer consists of Virtual Processors (VPs). These virtual VPs are connected using same network topology as the application model, however using the bounded size data channel components. The main purpose of the mapping layer is to forward the event traces from the Kahn process in the application layer to the the architectural components in the architecture model. This forwarding is done according to a user-specified mapping of application processes and communication channels onto processors and communication structures respectively. The components in the mapping layer simulate synchronization of communication events in such a way that forwarded events are "safe": that means they don't cause deadlock due to unmet data dependencies when mapped onto shared resources. In Sesame framework, the application model is not timed, while the mapping layer and architecture layer are modeled in (the same) timed simulation domain.

In the architecture model, the architectural timing consequences of the events are modeled. The processor components model the processor utilization of the application process by using a lookup table that related to each computation(EX) events to an execution. The interconnection and the memory components model the utilization and the contention caused by communication events - Read (R) and Write (W). These latency values may be obtained from literature, hardware measurements, rough estimates or

from more detailed simulators such as described in [10].

V. System Framework

Within the context of this research, we are focused on developing an approach for system level design space exploration and dynamic mapping for dynamically reconfigurable systems targeting the streaming applications of multimedia domains. The application is represented as a graph at granularity of the coarse-grain task or function level KPN (Kahn Process Network) [9] specification, KPN = (Vk, Ek), where set Vk and Ek refer to the Kahn nodes and the directed FIFO channels between these nodes, respectively. Kahn processes are functions which operate on streams. This streaming nature of Kahn processes makes KPNs very suitable for modeling the dynamic nature of streaming applications of the multimedia domains. For this particular reason, KPN is chosen as a modeling structure to model the application behavior. The KPN graphs used in this case are static and acyclic KPN graph. An example KPN graph is shown in Figure 4.

There are different types of tasks to be specified in the system: *fixed-software* tasks, *fixed-hardware* tasks and *pageable* tasks. The fixed-software tasks are those tasks which are implemented always on software, fixed-hardware tasks are those which are implemented only as hardware and pageable task are those which can be switched between hardware and software processor and can be executed on both of the resources. Given an acyclic KPN graph and given the different types of the tasks, the partitioning assigns each task to the given task set. And, given a partitioning, the mapping binds a given task to a processing resource.

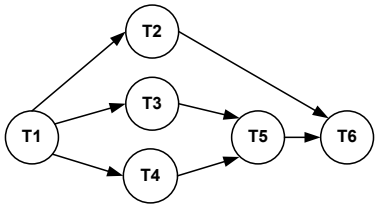

Fig. 4
Example KPN Graph of the Application

To illustrate this, consider the example application graph as in Figure 4 and a given reconfigurable architecture Molen. Molen consists of two types of processing elements GPP (a general purpose processor) and RP (a reconfigurable processor) [for detail description of the Molen architecture refer to Section III]. The partitioning identifies the list of the tasks as: *fixed-software*, *pageable* tasks and *fixed-hardware*

fixed-software tasks = $(T_1)_{SW}$
pageable tasks = $(T_2, T_3)_{HW/SW}$ and,
fixed-hardware tasks = $(T_4, T_5, T_6)_{HW}$

This process of identifying set of tasks that belongs to a particular resource type is called *spatial partitioning*. Given the above partitioning, the mapping binds each task to execute on a particular resource. For the spatial partitioning given above the following mappings can be identified.

Mapping 1 : $(T_1, T_2)_{GPP}$, $(T_3, T_4, T_5, T_6)_{RP}$
Mapping 2 : $(T_1, T_2, T_3)_{GPP}$, $(T_4, T_5, T_6)_{RP}$ etc.

A reconfigurable architecture is subject to various types of system constraints such as area, power etc. Due to the area constraint, the reconfigurable hardware can accelerate only as much of the program as it fits within the programmable structures. As a result, not all the tasks that are mapped onto RP can be executed on the RP at the same time. Therefore, depending on the area capacity of the RP, these tasks have to divided into various *configurations* such that each configuration can be executed onto the RP at once. This process is called *temporal partitioning*.

Let us assume the given RP cannot fit more than two tasks at once, in that case, execution of the above mappings on RP is not possible at once. As a result, the tasks mapped to RP have to be divided into different configurations as below:

Mapping 1 : $(T_1, T_2)_{GPP}$, $((T_3, T_4)_{C_1}, (T_5, T_6)_{C_2})_{RP}$;
Mapping 2 : $(T_1, T_2, T_3)_{GPP}$, $((T_4, T_5)_{C_1}, (T_6)_{C_2})_{RP}$;

where C_1 and C_2 are different configurations

In this case, in Mapping 1, though tasks T_5 and T_6 are in the same configurations, due to the dependency between these two tasks, they cannot run on RP at the same time. However, the task T_6 can be configured and made ready for execution on the RP while the task

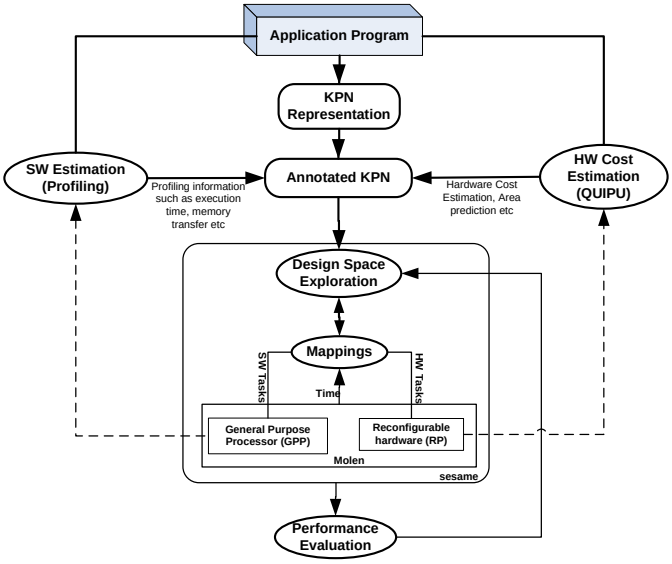

Fig. 5
THE EXPLORATION FRAMEWORK

T_5 is still executing. This overlaps the configuration time of one task with the execution time of another task. This is called configuration hiding. In similar ways, the configuration hiding is possible also for the tasks T_4, T_5 in Mapping 2.

From a given acyclic KPN graph, the goal of the exploration framework is to dynamically identify a set of hardware tasks and a set of software tasks, to allocate the architectural resources to these tasks, and to schedule these tasks such that all the application as well as architectural constraints are satisfied. For a dynamic system where application and architecture can evolve, these task sets can change at the runtime, as a result at the end of exploration three sets of task can be identified viz hardware task, software task and pageable task. This allows for a system to be optimized based on runtime behavior and values, which is hard to determine using any static methods.

Moreover, the designers can specify any kind of design constraints in the system such as area, communication, power consumption etc. The designers can also indicate various design objectives such as to maximize the performance of the system, to minimize the power consumption etc. The exploration framework takes these goals and the constraints into consideration and finds the optimum task sets for architectural mapping. The dynamic reconfiguration capabilities of the architecture is also taken into account, meaning, not only it finds the best tasks for implementation onto reconfigurable hardware, it also finds how different configurations can be mapped sequentially on reconfigurable hardware in order to share the available area.

The exploration framework is shown in Figure 5. In the first phase, the application program is transformed into KPN specification. The designer guides the identification and extraction of the critical regions and can also implicitly decide for some task to be either fixed-software, fixed-hardware and pageable. The granularity of task can also be specified in this phase. The assumption is that, each task in the KPN graph always has less or equal constraint than any constraints imposed on the RP. For instance, each task on KPN graph individually cannot take more area than total RP area available, which implies, $A_i \leq A_{RP}$ where A_i is the area required by task T_i and A_{RP} is total the RP area.

As a preprocessing phase, the application is profiled to identify performance critical regions (in terms of computation time and communication time). The profiling of the application provides various estimations for the task when it executes as software tasks on GPP. It analyzes the program statically and/or dynamically in order to determine relevant information such as execution time, memory size, number of time

the task is executed etc. In this context, we are mainly interested in following SW estimates:
1. **Computation Time** is the quantitative measure of the total execution time for a task while it is executing on GPP. This is the software execution time for a task (T_i) in a given KPN graph and is denoted as t_{iSWexe}.

2. **Communication Load** is the quantitative measure of the total number of bytes exchanged (written or read) through a FIFO channel in a given KPN graph. It is computed as product of number of tokens sent through a FIFO channel and the size of a token in bytes sent through that particular channel. Communication load between tasks T_i and T_j can be calculated as:
$CL_{ij} = \sum_{k=0}^{N} n_{ijk}.m_k$; where n_{ijk} is the number of token sent through channel k between tasks T_i and T_j, m_k is the token size of channel k and N is total number of FIFO channels between T_i and T_j.

Similarly, hardware cost estimations are provided by quantitative hardware estimation model (QUIPU model [11]). The hardware prediction method estimates different hardware attributes (such as hardware area, interconnect delays, hardware latency etc) for design exploration and partitioning. These are the estimated values for the task attributes when a task is executing on the RP. We are interested in following HW estimates:
1. **Area Occupancy** is the quantitative measure of the area occupied by a task on a given RP. This can be measured in slices and can also be expressed as a percentage of the total area of the RP. The area prediction for a task T_i is denoted by A_i

2. **Hardware Latency** is the quantitative measure of the total execution time for a task while it is executing on RP. The hardware execution time for a task (T_i) is denoted as t_{iHWexe}.

Every task in the KPN graph is annotated with these attributes. In this particular phase, designers can also apply several heuristics methods on the given annotated KPN graph to statically identify various mappings for the underlying architectures. The different mappings generated are then given to the architecture for execution. However, in the dynamic system where the application as well as architecture behavior changes, in order to satisfy changing behavior of the dynamic system, the design space exploration has to be performed at the runtime by identifying different set of mapping at different period of application execution. For this, the mapping of the tasks is altered from one resource to another resource at one period to another period. As a simple example, assume, at particular period t_1 tasks T_1, T_2 are mapped onto GPP and tasks T_4, T_5, T_6 are mapped onto CCU1, CCU2 and CCU3 of RP. Lets say at time period t_2, due to some reason CCU2 has been shut down in order to save power in the RP, as a result the mapping identified in time t_1 is not feasible in time period t_2. Hence, the mapping has to be changed dynamically by moving the tasks from one resource to another.

The exploration process is guided by the various information provided from the underlying architecture such as free resources, timing information etc. Based on these information about applications as well as architecture, mapping desision is made for every task. The performance impacts of each mapping is monitored for a particular system funnction on the given architecture. This evaluation results can assist for further decision making while identifying various other mappings.

VI. Summary and Future Work

To tackle the enormous design space resulted with the increasing functionalities and the massive use of reconfigurable heterogeneous platforms, it is necessary to perform Design Space Exploration at various design levels to explore the tradeoffs between various design goals and to find optimal solutions. Performing DSE at very early stages facilitates design decision to be made early, as a result overall design time can be significantly reduced. In the dynamic system where application as well as architecture behavior changes, in order to satisfy such changing behavior of the dynamic system, the design space exploration has to be performed at runtime to evaluate the design for runtime values and behaviors. In our future work, we will implement and validate the runtime exploration and mapping of the reconfigurable architectures for the dynamic systems.

References

[1] K. Compton and S. Hauck, "Reconfigurable computing: A survey of systems and software," *ACM Computing Surveys*, vol. 35, June 2002.

[2] T. Todman, G. Constantinides, S. Wilton, O. Mencer, W. Luk, and P. Cheung, "Reconfigurable computing: ar-

chitectures and design methods," in *IEEE proceedings of Computers and Digital Techniques*, vol. 152, June 2005, pp. 193–207.

[3] K. Sigdel, M. Thompson, A. Pimentel, T. P. Stefanov, and K. Bertels, "System-level design space exploration of dynamic reconfigurable architectures," in *Proceeding of International Symposium on Systems, Architectures, MOdeling and Simulation (SAMOS VIII Workshop), Samos, Greece*, July 2008, pp. 279–288.

[4] "Delft work bench: http://ce.et.tudelft.nl/dwb/."

[5] S. Vassiliadis, S. Wong, G. N. Gaydadjiev, K. Bertels, G. Kuzmanov, and E. M. Panainte, "The molen polymorphic processor," *IEEE Transactions on Computers*, pp. 1363–1375, November 2004.

[6] S. Vassiliadis, G. N. Gaydadjiev, K. Bertels, and E. M. Panainte, "The molen programming paradigm," in *Proceedings of the Third International Workshop on Systems, Architectures, Modeling, and Simulation*, July 2003, pp. 1–10.

[7] C. Erbas, A. D. Pimentel, M. Thompson, and S. Polstra, "A framework for system-level modeling and simulation of embedded systems architectures," *EURASIP J. Embedded Syst.*, vol. 2007, no. 1.

[8] A. D. Pimentel, C. Erbas, and S. Polstra, "A systematic approach to exploring embedded system architectures at multiple abstraction levels," *IEEE Trans. Comput.*, vol. 55, no. 2, pp. 99–112, 2006.

[9] G. Kahn, "The semantics of a simple language for parallel programming," in *Proc. of the IFIP Congress 74*, 1974.

[10] A. D. Pimentel, M. Thompson, S. Polstra, and C. Erbas, "On the calibration of abstract performance models for system-level design space exploration," in *ICSAMOS*, G. Gaydadjiev, C. J. Glossner, J. Takala, and S. Vassiliadis, Eds. IEEE, 2006, pp. 71–77.

[11] R. J. Meeuws, Y. D. Yankova, K. Bertels, G. N. Gaydadjiev, and S. Vassiliadis, "A quantitative prediction model for hardware/software partitioning," in *Proceedings of 17th International Conference on Field Programmable Logic and Applications (FPL07)*, August 2007, pp. 735–739.