Towards an ESL Design Framework for Adaptive and Fault-tolerant MPSoCs: MADNESS or not?

Emanuele Cannella*, Lorenzo Di Gregorio†, Leandro Fiorin‡, Menno Lindwer§, Paolo Meloni¶, Olaf Neugebauer‖, and Andy Pimentel**

* Leiden Institute of Advanced Computer Science, Leiden University, The Netherlands
Email: cannella@liacs.nl

† Lantiq Deutschland GmbH, Neubiberg, Germany
Email: lorenzo.digregorio@lantiq.com

‡ ALaRI, Faculty of Informatics, University of Lugano, Switzerland
Email: fiorin@alari.ch

§ Silicon Hive BV, High Tech Campus 83, 5656 AG Eindhoven, The Netherlands
Email: menno.lindwer@siliconhive.com

¶ DIEE - Department of Electrical and Electronic Engineering, University of Cagliari, Cagliari, Italy
Email: paolo.meloni@diee.unica.it

‖ Informatik Centrum Dortmund, Joseph-von-Fraunhofer-Str. 20, Dortmund, Germany
Email: neugebauer@icd.de

** Computer Systems Architecture Group, Informatics Institute, University of Amsterdam, The Netherlands
Email: a.d.pimentel@uva.nl

***Abstract*—The MADNESS project aims at the definition of innovative system-level design methodologies for embedded MPSoCs, extending the classic concept of design space exploration in multi-application domains to cope with high heterogeneity, technology scaling and system reliability.**

The main goal of the project is to provide a framework able to guide designers and researchers to the optimal composition of embedded MPSoC architectures, according to the requirements and the features of a given target application field. The proposed approach will tackle the new challenges, related to both architecture and design methodologies, arising with the technology scaling, the system reliability and the ever-growing computational needs of modern applications.

The methodologies proposed with this project act at different levels of the design flow, enhancing the state-of-the art with novel features in system-level synthesis, architectural evaluation and prototyping. Support for fault resilience and efficient adaptive runtime management is introduced at hardware and middleware level, and considered by the system-level synthesis as one of the optimization factors to be taken into account.

This paper presents the first stable results obtained in the MADNESS project, already demonstrating the effectiveness of the proposed methods.

I. Introduction

Modern embedded systems, which are increasingly based on Multi-Processor System-on-Chip (MPSoC) architectures, usually integrate components provided by different parties, very often exposing MPSoC designers to a high level of design complexity. This complexity originates from the need of fitting different computational tasks to specific kind of processing elements, achieving an optimal exploitation of the different types and degrees of parallelism available in the workload. Moreover, irregular communication patterns between the tasks often require the design of dedicated interconnect infrastructures, which optimally sustain communication among several subsystems with different characteristics. The resulting level of complexity exposes to the designer many degrees of freedom which require ad-hoc design methodologies to be managed for obtaining an efficient design [1]. The common practice in this area consists of traversing the system-level design space by means of an iterative process, known as Design Space Exploration (DSE), which evaluates and refines different candidate architectures to find an optimal solution.

To improve the overall productivity against chip level design flows, effective system-level design flows need to take into account, at early stages, a larger number of design variables as well as more and higher complexity IP cores inside the design space available to the designer. This need becomes particularly striking when multiple applications are executed simultaneously, unpredictably interfering at runtime with each other and demanding the guarantee of a given level of Quality of Service (QoS). For certain application classes, the existing "static" design of embedded processors needs to be extended by including "dynamic" generation of the processor IP to match configurable features with the demands of the prospective applications. In addition, given the increasing complexity of new systems, a certain degree of fault tolerance must be guaranteed, although constraints presented by embedded systems design make approaches involving massive redundancy hardly adoptable. These issues require the support inside modern MPSoCs of a certain degree of adaptivity while the designer must be able deal with challenges posed by modern technology nodes, relying on accurate estimations of the hardware costs

of each candidate architectural solution.

In this paper, we present an overview of the system-level design framework that is currently being developed within the MADNESS project. Given the requirements and the features of the targeted multimedia application field, this framework is able to guide designers and researchers to the optimal composition of an MPSoC architecture based on a Network-on-Chip (NoC). Its aim is to tackle several new challenges, related to both architecture and design methodologies, arising with the technology scaling, the system reliability and the ever-growing computational needs of modern applications.

In order to improve design predictability, the project aims at advancing static analysis techniques in order to include system behavior, and to exploit detailed and reliable emulation, based on FPGA prototyping when evaluating different candidate system configurations. When analyzing the emulation results, variables strictly related to a prospective implementation phase are taken into account, introducing "technology awareness" inside the system level design process and improving the "convergence", i.e. the accuracy of the results that can be predicted during the system-level design phase against those eventually obtained after the on-silicon implementation.

Furthermore, this framework also considers continued availability of service, in addition to more traditional constraints (typically, cost, performance, power consumption). Thus, it accounts for fault recovery as one of the optimization factors to be satisfied. As a consequence, it supports adaptive runtime management techniques, tailoring the architectures under new metrics posed by novel dynamic strategies and advanced support for related communication issues.

The remainder of this paper is organized as follows. The next section provides a brief overview of the MADNESS framework, after which the subsequent sections provide more details about the different parts of the framework. In section III, the simulative DSE component of the framework will be discussed. In Section IV, the FPGA-based evaluation platform is described. Section V, provides more information on the compilation toolchain and the hardware abstraction layer used in the framework. Section VI discusses the support for system adaptivity, after which Section VII describes the fault tolerance issues addressed by the MADNESS framework. Finally, Section VIII presents our conclusions.

II. The MADNESS framework: optimizing heterogeneous platforms

In figure 1, a block diagram of the MADNESS system-level design framework is presented. The framework aims at efficiently and effectively performing design space exploration (DSE) to search for the optimal composition of a multimedia NoC-based MPSoC architecture, operating on a library of heterogeneous industrial-strength IP cores and exposing a large number of degrees of freedom.

One of the main differences between the MADNESS project and other project oriented toward application mapping is that MADNESS' target platform is not static. Rather, the target platform consists of a library of IP blocks, mentioned in figure 1 as *Hardware Library*, which are explored in continuously varying configurations. The MADNESS project employs a variety of IP building blocks, among which are application-specific instruction-set processors (ASIPs), memories, interconnects, and adapters. This hardware IP library includes industrial-strength blocks from Silicon Hive and Lantiq. Some further blocks, such as a video motion processor, are specifically developed for the MADNESS project. Each architecture configuration is an abstract collection of target IPs and interconnect structures. The decisions during the optimization process are actually taken by a DSE engine, represented by the box at the top of figure 1 and described in detail in section III. The DSE engine iteratively selects one among multiple sets of architecture instances and application mappings, exploiting a specific layer for rapid and accurate architectural evaluation. The DSE engine is based on enhanced versions of several key elements from the Daedalus system-level synthesis design flow [2]. More specifically, it deploys the Sesame simulation environment [3] for simulative DSE and uses the PNgen/ESPAM tools for parallel application code generation [4]. As a consequence, the MADNESS framework, like Daedalus, uses Kahn Process Networks (KPNs) [5] to model parallel multimedia applications. As described later in deeper detail, the Sesame simulation environment has been extended in the scope of the MADNESS project to support DSE for MPSoCs sustaining multi-application dynamic workloads as well as to include with novel techniques for design space pruning including fault tolerance aspects. As a further point of novelty, the project continuously develops an evaluation layer which integrates a system-level synthesis flow to rapidly evaluate selected design points using an FPGA-based emulation and evaluation platform, described in further detail in section IV. The framework allows to perform system optimization by means of adequately interleaving a high-level simulative design space exploration process with the evaluation of selected design points synthesized on real FPGA-based prototypes. Thus, the DSE engine can access an FPGA-based environment for on-hardware prototyping, when needed during the optimization process, in order to obtain a detailed evaluation of a candidate architecture by actually executing the target application on the implemented prototype.

In order to improve design predictability, the flow is improved by annotating the emulation results on adequate analytic "technology-aware" models (energy consumption, execution time per frequency, area obstruction). This allows to translate emulation results to a reliable evaluation of a prospective ASIC implementation of the system on a given technology, before actually performing all the effort-hungry back-end fabrication steps.

In order to allow the execution of the target application on the FPGA implementation of completely different design points, featuring different kinds of processing elements and interconnects, the framework includes a re-configurable compilation toolchain, depicted in figure 1, which is capable of re-targeting itself according to the design point specification. Furthermore, a hardware abstraction layer (HAL) exposes to

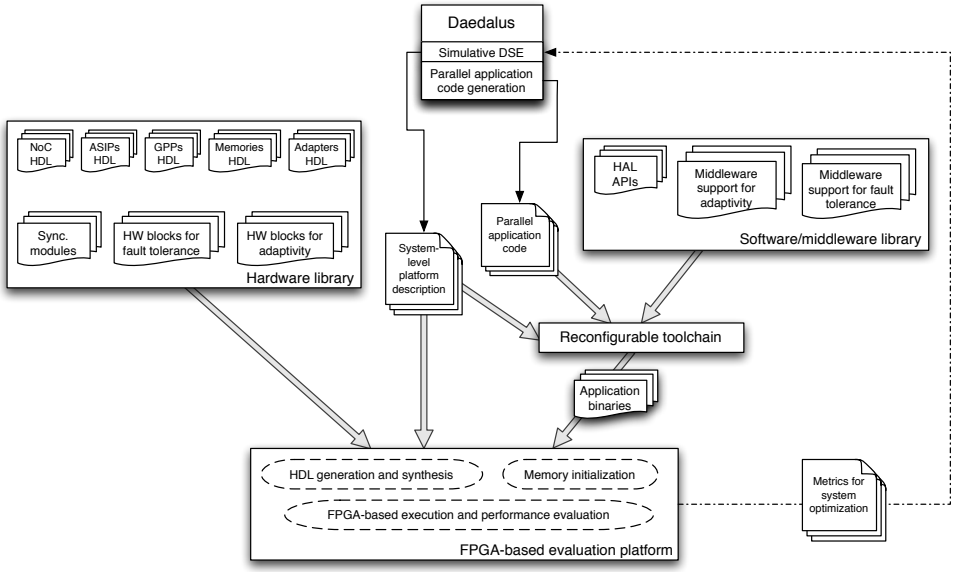

Fig. 1. The MADNESS system-level design framework for adaptive and fault-tolerant MPSoCs.

the programmer a convenient set of APIs that can be used to program the system without referring to specific low-level details of the platform. The compilation toolchain automatically links the appropriate implementation of the API included in the HAL, according to the description of the design point under evaluation. The compilation toolchain and the HAL are described in further detail in section V.

Support for system adaptivity and fault tolerance, described in sections VI and VII, has been introduced in the hardware library by new or modified IPs and in the middleware by a layer in charge of dynamically managing the system at runtime. The implementation is biased toward low redundancy and power consumption in order to meed the demands of the embedded systems domain. Characteristics of the dynamic behavior and of the resilience to faults can be taken into account by the system-level synthesis during the architectural optimization process, exploiting the mentioned extensions to Sesame.

To allow the mentioned tools and methods to inter-operate without or with minimal manual intervention, the IP-XACT standard [6] was selected for exchanging abstract platform instance descriptions between different tools. However, with regard to this purpose, IP-XACT has shown a number of shortcomings. As a result, several adaptations were made to the IP-XACT standard, allowing us to capture the variability of the target architectures and to capture the power and area consequences of DSE design choices. Finally, in order to actually construct the heterogeneous platform instances needed for FPGA-based evaluation, the MADNESS project has resulted in novel approaches to automatically convert MADNESS IP-XACT descriptions into RTL implementations of multi-ASIP platform instances.

III. Simulative DSE

The MADNESS framework deploys the Sesame MPSoC simulation framework [3] for simulative DSE. Sesame recognizes separate application and architecture models within a system simulation. An application model, specified as a KPN, describes the functional behavior of a (set of) concurrent application(s). An architecture model defines architecture resources and captures their performance constraints and power consumption characteristics. Subsequently, using a mapping model, an application model is explicitly mapped onto an architecture model (i.e., the mapping specifies which application tasks and communications are performed by which architectural resources in an MPSoC), after which the application and architecture models are co-simulated to study the performance and power consumption consequences of the chosen mapping.

To actually search the design space for optimum design points, Sesame utilizes heuristic search techniques, such as multi-objective Genetic Algorithms (GAs). Such GAs prune the design space by only performing a finite number of design-point evaluations during the search, evaluating a population of design points (solutions) over several iterations, called generations. With the help of genetic operators, a GA progresses iteratively towards the best possible solutions.

A. Scenario-based DSE

Thus far, Sesame's DSE was focused on the analysis of MPSoC architectures under a single, static application workload. The current trend, however, is that application workloads executing on embedded systems become more and more dynamic. Not only is the behavior of a single application changing over time, but the effect of the interactions between different applications are also hard to predict. This dynamic behavior can be classified and captured using so-called workload scenarios [7]. Workload scenarios make a distinction between two

aspects. First, intra-application scenarios describe the dynamic behavior within applications. For example, a QoS mechanism within a decoder application may dynamically lower the bit-rate to save power while still meeting its deadlines. Second, inter-application scenarios describe the interaction between different applications that are concurrently executing on an embedded system and contending for its system resources.

In the context of MADNESS, we have developed a novel scenario-based DSE method that allows for capturing the dynamic behavior of multi-application workloads in the process of system-level DSE [8]. An important problem that needs to be solved by such scenario-based DSE is the rapid evaluation of MPSoC design instances during the search through the MPSoC design space. Because the number of different workload scenarios can be immense, it is infeasible to rapidly evaluate an MPSoC design instance during DSE by exhaustively analyzing (e.g., via simulation) all possible workload scenarios for that particular design point. As a solution, a representative subset of workload scenarios can be used to make the evaluation of MPSoC design instances as fast as possible. The difficulty is that the representativeness of a subset of workload scenarios is dependent on the target MPSoC architecture. But since the evaluated MPSoC architectures are not fixed during the process of DSE, we need to simultaneously co-explore the MPSoC design space and the workload scenario space to find representative subsets of workload scenarios for those MPSoC design instances that need to be evaluated. To this end, we have developed a scenario-based DSE method combining a multi-objective GA and a feature selection algorithm. The GA is used to search the MPSoC design space, while the feature selection algorithm dynamically selects a representative subset of scenarios. This representative subset of scenarios is then used to predict the quality of MPSoC design instances in the GA as accurately as possible.

This scenario-based DSE is depicted in figure 2. As input, the scenario-based DSE uses the application models that need to be mapped onto the MPSoC, an MPSoC platform model, a scenario database in which all possible application scenarios are stored, and search parameters. As output, the DSE produces candidate MPSoC design instances that perform well when considering all the potential situations that can occur in the specified dynamic multi-application workload.

B. Improving the DSE with domain-knowledge

For the mapping DSE performed by Sesame, a vector-based mapping specification – specifying which application task or communication channel is mapped onto which architectural MPSoC resource – is quite useful (for example, a genetic algorithm can use the vector as the genotype for a population element). However, multiple specifications can describe the same mapping. For example, mapping A : $\{1,0,1,2\}$ and B : $\{2,0,2,1\}$ (where the *i*-th index indicates the target processor of application task *i*) denote the same (duplicate) partitioning, and therefore an equivalent mapping on a homogeneous platform. This can be solved by using a normal form notation of mappings, but here we introduce a *mapping distance metric*,

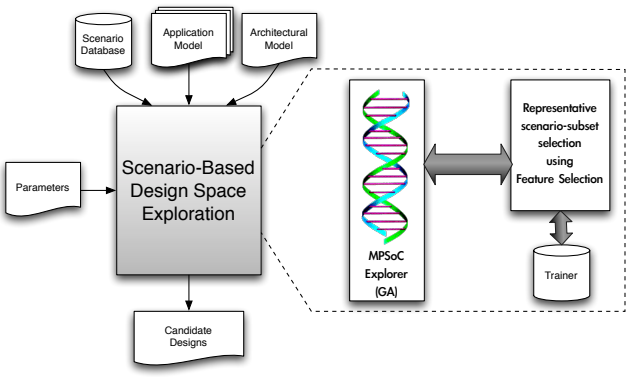

Fig. 2. Scenario-based DSE.

δ, which can distinguish duplicate mappings and which is a useful concept for improving the effectiveness of the GA-based DSE (as will be explained below). The mapping distance $\delta(p,q)$ is defined as the minimum number of task re-mappings that is required to transform mapping p into mapping q. For example, $\delta(A,B) = 0$ and $\delta(A,C) = 2$ if C : $\{0,1,2,1\}$. For all duplicate mappings p and q, $\delta(p,q) = 0$. Note that with minor modifications, the distance metric can also be defined for heterogeneous platforms (where duplicate mappings can still occur on groups of processors with the same type).

With GAs (and evolutionary algorithms in general), there is a delicate balance between convergence and population diversity: some instances of GAs converge too quickly, while others do not converge at all (or not sufficiently). In the former case the achieved result is often a local optimum: the population is insufficiently diverse due to selective pressure, so that subsequent iterations of the GA do not introduce new individuals in the population. On the other hand, when diversity is too high, GAs do not converge sufficiently, causing a kind of random walk through the search space. Diversity of a GA is partially regulated by the rate of mutation of individuals.

To illustrate how the mapping distance can improve the DSE, we show how it can improve the diversity within a GA by untangling individuals in the population that are clustered (too) close together. To this end, we use the distance metric to identify the one population individual that (on average) has the shortest distance to the other individuals in the population. Subsequently, we remove this individual and replace it by a new randomly generated individual. The experiment involves a case-study where the design space consists of an 11-process application mapped onto the 4-processor MPSoC. The optimum design point is known (by exhaustive search) for this particular design space. The results are shown in Figure 3. On the x-axis, the quality of the search result is shown as a percentile (a lower percentile indicates a result closer to the optimum) while the y-axis shows the probability of achieving a result with that quality. The red curve (bottom curve) shows the results for random search, the green curve (middle curve) refers to the results of an unmodified GA, and the blue curve (top curve) shows the results of the GA in which the mapping distance is used to improve diversity. Evidently, the mapping

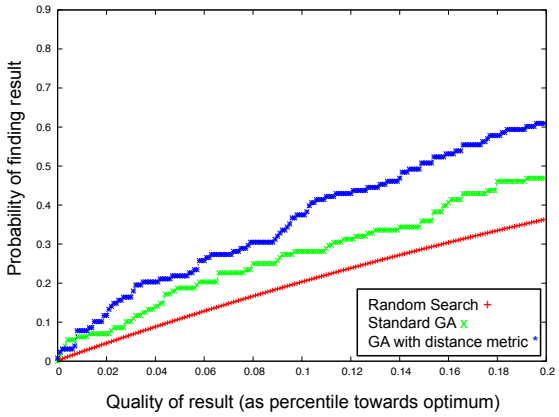

Fig. 3. Improving DSE using the mapping distance metric.

distance can help the GA to find better solutions.

IV. FPGA-BASED EVALUATION ENVIRONMENT

The design flow envisioned in MADNESS raises the need for a fast and accurate evaluation environment. Such a tool has to provide the upper layers of the design flow (i.e., the simulative DSE) with feedback about the performance of any requested candidate architectural configuration. To this aim, within the project, a flexible and fast FPGA-based emulation framework extending the work presented in [9] has been developed. It leverages a library of components, instantiates the desired system configuration, specified through a system-level specification file, and generates the hardware description files for the FPGA synthesis and implementation, automating design steps that are usually very error-prone and effort-hungry. The mentioned feedback, namely consisting of detailed and precise event/cycle-based metrics, is obtained from the execution of the target software application (compiled and linked with the proper toolchains and communication libraries) on the candidate system configuration, implemented on FPGA and adequately instrumented with counters and hardware probes. Moreover, the prototyping environment provides support for *"technology awareness"* within the DSE process, by coupling the use of analytic models and FPGA-based fast emulation. This allows to obtain early power and execution time figures related to a prospective ASIC implementation, without the need to perform long post-synthesis software simulations. The FPGA emulation results are back-annotated using analytic models for the estimation of the physical figures of interest. Timing results (cycle counts) are evaluated according to the modeled target ASIC operating frequencies and the evaluated switching activity is translated into detailed power numbers. Thus, the assumptions made at the system-level design phase can be verified before the actual back-end implementation of the system, increasing the overall convergence of the design flow. The models included in the evaluation platform are built by interpolation of layout-level experimental results obtained after the ASIC implementation of the reference library IPs, along the lines already defined for NoC building blocks in [10]. In the latter, the accuracy of the models is assessed to be higher than 90% when complete topologies are considered, with respect to post-layout analysis of real ASIC implementations.

A. Prototyping speed-up techniques

In those exploring iterations that require a detailed estimation of the functional and physical features of the candidate system-level configuration, emulation on FPGA is much faster than low-level software-based simulation, especially when complex systems are targeted. However, the achievable speed-up is counterbalanced by the computational effort related to the synthesis and implementation flow. In order to push on-hardware prototyping one step further, we built tools that implement on the FPGA a system configuration that is over-dimensioned with the hardware resources necessary to emulate all the network topologies and processor configurations included in a predefined set of design candidates. Then, at runtime, each specific candidate design point is mapped on top of the implemented hardware, exploiting dedicated software-based configuration mechanisms. In this way, several emulation steps can be performed after a single FPGA synthesis and implementation run, resulting in a speed-up of the whole topology selection process. The same kind of approach has been applied at interconnection-level and at processor-level, enriching two HDL generators included in the prototyping environment, respectively in charge of creating the synthesizable description of the network-on-chip topology and of the processors instantiated in the system.

B. Fast NoC topology selection

Given a set of interconnection topologies to be emulated, we identify and reconfigure what we call a worst case topology (WCT). Specific hardware-software mechanisms have been implemented to enable the possibility of mapping the topology under test on top of the WCT, reconfiguring the connections to avoid accounting for latencies introduced by the switching elements that are not included in the topology to be emulated. Establishing a direct zero-latency connection of two specific ports of a generic NoC router has been made configurable at runtime, by means of dedicated low-level C functions automatically created by the HDL generator, resulting in the creation of a combinational path bypassing a certain switch. To this end, the hardware of the baseline reference NoC router has been enriched to support runtime reconfiguration of the routing strategy.

C. Fast ASIP configuration selection

The mentioned approach has been applied to the emulation of Silicon Hive's VLIW ASIPs. We provide the capability of prototyping several instances of the Processor Architecture Template on the same FPGA implementation. Each design point is a different composition of substructures called Template Building Blocks, namely issue slots (vertical datapaths slices through the Processor Template, representing VLIW ways), register files, function units (the hardware blocks implementing the op-codes), logical memories and internal interconnect structures (implementing the connectivity within

the processor). The enhanced HDL generation utility analyzes the whole set of configurations under prototyping, synthesizes the worst case configuration (WCC), and creates the configurable hardware and the software functions needed to map each candidate configuration on top of the over-dimensioned hardware. An instruction adapter is inserted in the processor template to enable the execution of binaries compiled for a given configuration under test on the WCC, taking care of adequately manipulating and dispatching the instruction fields through the over-dimensioned datapath.

The tables in figure 4 and 5 illustrate the impact of providing fast reconfiguration for both ASIP and FPGA flows in terms of FPGA device occupation and critical path degradation. With respect to the ASIP DSE, data presented refers to implementation of the largest configuration found in a pool of 16 candidates versus the WCC supporting reconfiguration for each candidate of the same pool. With respect to the NoC topology selection, we report results obtained for two DSE runs involving respectively 4 (DSE A) and 16 (DSE B) topologies under prototyping (TUP in the figure).

	Occupied Slices	**Slice Registers**	**SLice LUTS**
Largest TUP (DSE A)	17327(33%)	33,885(16%)	44673(21%)
WC (DSE A)	20627(39%)	41313(19%)	58862(28%)
Largest TUP (DSE B)	17397(33%)	34487(16%)	44926(21%)
WC (DSE B)	21815(42%)	44943(21%)	64696(31%)

	Critical path
Slowest TUP (DSE A)	10,902
WC (DSE A)	10,902
Slowest TUP (DSE B)	10,976 ns
WC (DSE B)	11,307 ns

Fig. 4. Experimental results related with the hardware overhead introduced by the support for fast NoC prototyping

	Occupied Slices	**Slice Registers**	**Slice LUTS**
WCC	21278(41%)	6931(3%)	17951(8%)
Largest ASIP	19859(38%)	6923(3%)	16387(7%)

	Critical path
WCC	9.817*ns*
Slowest ASIP	9.809*ns*

Fig. 5. Experimental results related with the hardware overhead introduced by the support for fast ASIP prototyping

As may be noticed, the introduced device utilization overhead is always limited and is controllable when the number of candidate configurations increases. Results also show how the critical path is almost insensitive to the introduction of the support for rapid prototyping. Thus, the overhead reduction mechanisms effectively allow accelerating the prototyping of MPSoCs, without having a significant impact on the system size and complexity that can be managed by the prototyping platform. It was possible to implement systems including 32 RISC cores on commercial devices, showing the approach to be compliant with significant system sizes. Future work will involve the analysis of multi-FPGA techniques to further increase the scalability of the prototyping method.

V. Compilation toolchain and hardware abstraction layer

This section describes the interaction between the DSE, the compilation toolchain and the evaluation platform. Further, a brief overview of the extensions to the available compilers is given. Finally the key features of the hardware abstraction layer and their integration in the compilation toolchain are described.

A. Compilation toolchain

The DSE tool passes the sources for each process and additional information like mapping and system description to the compilation toolchain (see Figure 1). The compilation toolchain takes care of the correct mapping between processes and their corresponding compilers for each processor in an automatic way. In order to determine the right compiler, the user needs to pass an environment description to the compilation toolchain. This description defines which compilers are available and which options to use for each processor in the system.

To meet the requirements of the framework, a hardware abstraction layer (HAL) is integrated in the compilation toolchain in a retargetable manner.

B. Hardware abstraction layer

The MADNESS framework aims at generating an optimal MPSoC for given hardware components and for a given application. Therefore, the application developer does not know the platform during the development process. Special processor-dependent instructions cannot be exploited during system generation. Even the realization of simple low-level functions like communication or synchronization without knowledge of the underlying processors and hardware components is impossible. For this reason, the MADNESS framework provides a Hardware Abstraction Layer (HAL) which enables the developer to create portable and processor-dependent optimized applications. The presented HAL consists of two parts, a fixed part which covers the standard abilities of processors and a generic processor-dependent part.

1) Standard abilities: Standard abilities include memory access, communication or synchronization mechanisms. One assumes that present and future processors support these mechanisms. As a consequence MADNESS defines a fixed set of standard functions which have to be implemented for each available processor used in this framework.

2) Special abilities: The actual advantage of heterogeneous multi-processor-systems is the composition of different specialized processors. To achieve the best performance one has to take into account that the HAL has also to cover the processor-dependent features. These features are called special ability functions. This part of the HAL is generic and extensible by the user without modifications of the used compilers. For each specialized implementation, a standard ANSI-C implementation must be provided. Thus, the availability of a semantically

equivalent application, if the corresponding processor feature is not available, is ensured.

3) HAL library: A special HAL library provides the processor-dependent implementations. This library is integrated in the compilation toolchain and enables compilers to generate binaries for each processor. Additionally, new HAL functions can be easily integrated without any changes to existing tools.

VI. Support for System Adaptivity

The term system adaptivity refers to the ability of a system to dynamically assign tasks of the application(s) running on it to the resources available over time. This is an emerging topic in MPSoC design due to recent evolutions in embedded systems [11]. The main reasons for this trend are:

- applications are getting intrinsically dynamic: for instance, a streaming application can lower its resolution or its frame rate if the battery charge is running low;
- modern systems are open to new incoming applications, which can not be analysed at design time: the workload on such systems cannot be predicted;
- technology scaling below the 32-nm node is leading to components which are more prone to temporal or even permanent faults: in case of a malfunctioning system component, the rest of the system should take over its tasks (also see the next section).

The KPN model of computation (MoC), adopted in the MADNESS framework to model multimedia applications for mapping them onto the MPSoC, presents remarkably simple operational semantics and distributed control, which allow for a natural realization of system adaptivity mechanisms.

From an architectural point of view, the framework is focused on tile-based NoC [12] systems. Among other advantages, this choice is driven by the goal of system adaptivity. NoC-based interconnects' flexibility allows to overcome the drawbacks exposed by point-to-point connections classically used in multimedia. Point-to-point connections are typically more efficient in terms of communication latency, but they are intrinsically less efficient in supporting communication patterns varying at runtime, unless full connectivity among all the processors in the system is provided at design time, at the price of making wiring and buffering of the whole communication structure rather complex. Moreover, NoC communication infrastructures are physically and functionally more scalable than shared-bus-based systems [13].

The starting assumption is that the target platform is equipped with a heterogeneous set of cores interconnected with a NoC. The application code must remain the same in every possible mapping of the tasks to allow for system adaptivity. This fact implies that the used communication primitives must be neither platform dependent nor mapping dependent. An intermediate layer, or *middleware*, has been implemented to refine such communication primitives, including mapping related information, and to respect the KPN semantics on the NoC-based MPSoC platform.

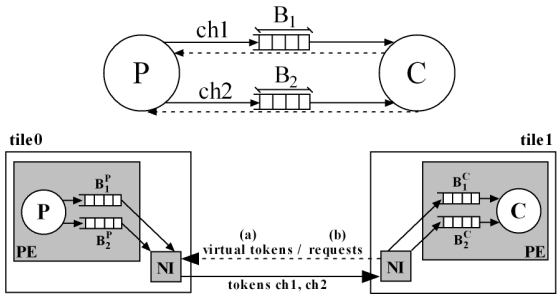

Fig. 6. Producer-consumer implementation: when using the VC and VRVC, the producer receives back virtual tokens (a); when using R, it receives requests (b).

A. Implementation of KPN semantics on NoCs

NoCs allow for system adaptivity but do not naturally match the KPN semantics. In order to implement the KPN semantics on our NoC-based platform, a flow-control policy must be put in place. We propose the distribution of the KPN's FIFO buffer(s) on the producer and consumer sides, as shown in figure 6. KPN processes communicate and synchronize using these FIFO buffers, i.e. they must comply with the *blocking read* and, in the special case of KPNs that we use (namely, Polyhedral Process Networks [14]) *blocking write* behavior. "Blocking read" means that a KPN process cannot execute if there are no data tokens on its input FIFO. "Blocking write" means that a process cannot write to a full output FIFO buffer.

Implementing the *blocking write* mechanism is challenging in the considered system, since we want to target generic NoC platforms, with no remote memory access. In this context, the producer cannot directly access the status of the consumer software FIFO. The basic requirement is that the producer cannot send tokens to the consumer over the NoC if the remote software FIFOs are full. Several approaches to guarantee this behavior have been implemented and compared, which are described below:

- **Virtual Connector (VC):** in this approach, for each channel of the KPN graph a virtual connector is instantiated in the opposite direction (refer again to figure 6). The consumer after reading one token sends back a virtual token to the producer over the virtual connector. In this way, the producer is acknowledged about the status of the consumer FIFO buffers.
- **Virtual Connector with Variable Rate (VRVC):** this approach differs from VC because the consumer can send virtual tokens less frequently, namely after *n* consumed tokens, with *n* that can be set from 1 to the size of the consumer FIFO buffer.
- **Request-based (R):** in this case the producer stores the output data token in its local FIFO, then sends all the available tokens, in a packet, upon receiving a request from the consumer. This approach has been previously proposed in [15], targeting the Cell BE platform.

We implemented and evaluated all the approaches listed above on a 2-by-2 NoC platform, using as case studies two

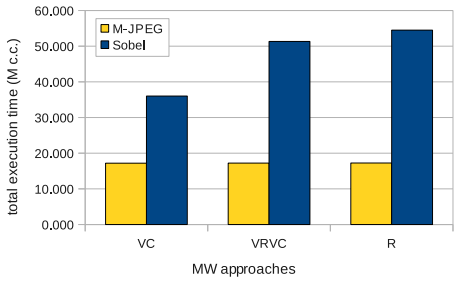

Fig. 7. Total execution time for different MW approaches.

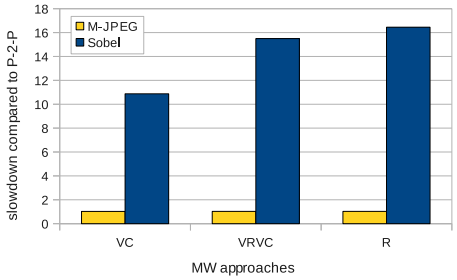

Fig. 8. Slowdown for different MW approaches.

applications which show extremely different characteristics. The first one is the Sobel filter, which is communication dominant and has a complex KPN graph topology. The second is an M-JPEG encoder, in which the computation/communication ratio is much higher and the KPN graph topology is fairly simple. The results are shown in figure 7: for the communication-dominant application (i.e. Sobel), the *VC* approach shows better results, while in the M-JPEG case all of the approaches show similar performances.

Figure 8 shows the comparison between the results obtained on the target NoC platform, endowed with the proposed middleware, and those obtained from custom point-to-point platforms generated by the ESPAM tool [4]. In the systems generated by ESPAM, a separate hardware FIFO is instantiated for each channel of the KPN graph, leading to a much more efficient communication because the platform matches the KPN MoC semantics. It is noticeable that the price paid for adaptivity and generality is quite high for communication dominant applications, while it is almost negligible when the computation/communication ratio is higher, as in the M-JPEG case study. The reasons why the communication onto the NoC platform is less efficient are mainly twofold. The first reason is that in this implementation, several KPN channels have to share the same physical channel (the NoC link). The second reason is a consequence of the first one. In the NoC case, the presence of only one physical link, being shared between different KPN channels, poses the need for a flow-control policy. To optimize for low hardware overhead, we chose to implement the control flow at the middleware level, based on software FIFOs on the producer and on the consumer side. This requires additional memory copy operations to dispatch/multiplex the communication tokens to/from the correct software FIFO. Such copies are unnecessary in the case of adoption of multiple point-to-point connections with hardware FIFOs. The significant overhead, which can be encountered when mapping some kind of applications on a NoC, confirms the need for a design optimization. As envisioned in the MADNESS project, this optimization shall be done using system-level design methodologies able to tailor the mapping and the architecture to limit as much as possible the mentioned overhead, while allowing for system adaptivity.

B. Middleware support for task migration

The middleware that was developed is capable of supporting all possible mappings, since it is able to refine the generic KPN primitives used in the KPN processes to mapping-dependent primitives. This refinement process makes use of *middleware tables* which include, among other useful information, the source and destination tiles' ID for each channel of the KPN. A variety of mappings of the M-JPEG application were successfully tested by replicating KPN tasks on different MPSoC cores and by activating the tasks on the desired cores and changing the middleware tables accordingly. Performing this migration process at run-time currently is ongoing work.

VII. Fault tolerance

Applications of embedded systems increasingly require high availability of the systems themselves, possibly accepting a measure of graceful degradation. Moreover, increasing complexity of the systems is reaching such levels that the probability that some manufacturing defect will escape end-of-production testing or that faults will become evident during normal operation has to be taken into account. Standard approaches based on massive redundancy are not directly applicable to embedded platforms, constrained by the need for solutions having low cost and low power consumption. New approaches are therefore needed.

The MADNESS project focuses on the development of fault tolerant solutions which are not dependent on a technology-related low-level fault model, but rather on technology-abstracting functional-level error models. This approach allows the development of a functionally identical system for two different implementation technologies - FPGA and ASIC - such that the system's evaluation on one technology be immediately adoptable and credible for the other technology.

The fault tolerant approaches implemented focus on the detection of run-time faults and on the use of reconfiguration strategies at different levels. In the MADNESS framework, three main types of components are taken into account, i.e., *processing cores*, *storage elements*, and the *network-on-chip* (NoC). While for storage components standard fault tolerant strategies based on error detecting and correcting codes are adopted, for the NoC and the processing elements ad-hoc strategies for fault detection and reconfiguration are being developed.

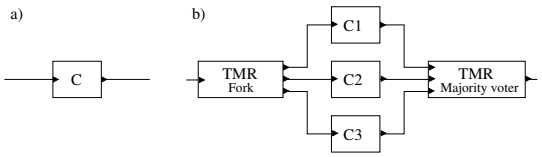

Fig. 9. Triple modular redundancy adaptation pattern applied at KPN level. KPN task C in (a) is replicated by three as shown in (b)

A. Fault detection

Detection of temporary and permanent run-time faults is performed at two different levels. At architectural level, self-checking strategies mainly based on error detecting and correcting codes have been implemented for NoC's components. At the detection of permanent faults, the reconfiguration of the communication system is initiated (as presented in Section VII-B).

In the case of the processing elements, detection of faults is performed at two levels. For non-critical applications, pre-designed software testing routines are employed [16]. Software routines are scheduled depending on the probability of having hard errors in the processors, allowing therefore the detection of permanent faults in the processing element or, depending on the granularity of their microarchitecture, in part of it. For this case, we allow the possibility of a (pre-defined) limited error propagation. In the case of critical applications, concurrent self-checking techniques have been implemented at KPN application level, by transforming the original KPN task graph accordingly to different levels of task redundancy [17]. For example, Figure 9 shows the case of the *triple modular redundancy* pattern, in which three parallel instances of the KPN component are created on different cores along with a module performing a fork on the task and a majority voter component. The fork operation creates a copy of the incoming message for each redundant instance and forwards each copy to the input ports of those instances. The majority voter component reads a token from all of its input ports and finds out the most recurrent token and sends it to its output connector, as well as signaling the core producing a faulty token, if detected.

B. Reconfiguration strategies

Reconfiguration is implemented based on both architectural-level design and on application-driven aspects.

1) Architectural-level strategy: For the *NoC*, whose correct behavior is essential for supporting higher level reconfiguration strategies, an architectural-level reconfiguration is employed. The on-chip communication network plays the important role of supporting the exchanges of information between nodes, and continuity of service is needed to guarantee the possibility of executing correctly applications and high level system services, such as system adaptivity and reconfiguration.

In order to be able to protect the NoC's elements (i.e., network interfaces (NIs) routers (or switches) and links) against "soft" and "hard" faults, the architecture of their main composing blocks (FIFOs, lookup tables, arbiters, etc) has been re-

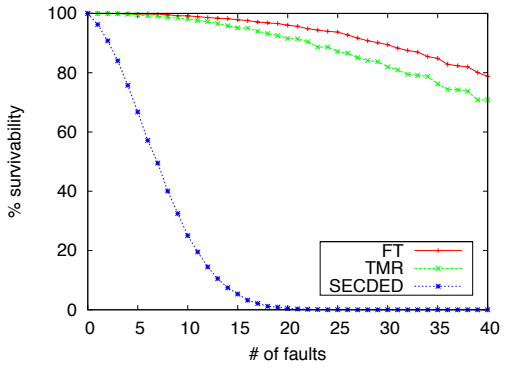

Fig. 10. Survivability of the NI by varying the number of errors injected

designed by employing a combination of error detecting and correcting code (ECC) and a limited architectural redundancy. Codes are useful for detecting and correcting soft errors as well as for identifying, by applying specific self-checking policies, permanent faults in the components, initiating therefore their reconfiguration for using the provided architectural redundancy. For example, Figure 10 compares the survivability of the NI developed (FT) with those obtained using a standard triple modular redundancy implementation (TMR), and one using only detecting and correcting codes (SECDED) [18]. The solution implemented provides a survivability comparable to the one obtained with *TMR*, while saving up to around 80% in area and energy consumption with respect to it.

2) Application-level strategy: In order to tolerate permanent failures in processing elements, and allowing a graceful degradation of the system's performance, the approach proposed in the MADNESS project focuses on the online remapping of the KPN tasks running on faulty processors. Task reallocation represents an obliged choice in particular in embedded systems, which can exploit the intrinsic availability of spare computation resources in modern MPSoC platforms.

In order to find an optimal solution to mapping tasks onto heterogeneous NoC multiprocessor systems, a method was developed which, starting from a task graph, provides as output the mapping of tasks onto tiles of the architecture. The method is based on an integer linear programming (ILP) formulation of the problem which is able to guarantee to find an optimal solution, in term of minimum communication cost and total computational time. Since the ILP solution is not scalable, and the needed execution time will increase significantly with the dimension of the NoC, the optimal solution also serves as a good reference point to be used when comparing the performances of heuristics-based solutions. Several heuristics were proposed for the remapping of tasks in the presence of run-time faults in different processing cores, obtaining a performance degradation within 6% with respect to the optimal remapping [19]. In order to support the migration of the tasks, a hardware module integrated within the NI of each NoC's node is being implemented. In fact, differently from the case of task migration for performance reasons in adaptive systems, faulty processors would be unable to implement the migration

procedure, and an external fault tolerant module is needed to support it.

VIII. Conclusions

In this paper, we presented the MADNESS system-level design framework, together with some first stable results obtained from the enrichment of a pre-existing system-level synthesis tool with novel features. These results have been obtained from the introduction of a concept for scenario-based exploration and novel distance-based design pruning techniques. The scenario-based exploration enables to consider variable-workload applications, multi-application workloads and faulty working conditions during the high-level synthesis. The novel distance-based design pruning techniques improve the search for optimal design points in the vast design space composed by heterogeneous and NoC-based MPSoCs. A significant speed-up of the accurate design point evaluation, required for tuning high-level simulation models, has been obtained within the mentioned system-level synthesis tool by developing interfaces toward an FPGA-based evaluation environment. The prototyping platform drastically reduces the overhead in terms of emulation time due to the traversing of the FPGA synthesis and implementation flow. Such features, together with the use of a re-targeting compilation toolchain, seamlessly reconfiguring itself to compile the target application for different processors within each design point under prototyping, enables the full exploitation of FPGA-prototyping within the system-level DSE process. Hardware-middleware strategies providing support for fault tolerance and adaptive runtime management of the architecture resources have been developed and successfully tested. Such strategies effectively take profit from the flexibility available in NoC-based communication structures and enable architectural DSE driven by the requirements of different workload and fault scenarios, allowing coexistence of different mapping configurations on the architecture. The developed flow-control introduces, on the other hand, a latency penalty that must be carefully controlled and minimized. This confirms the need and the usefulness of an optimization process able to take into account support for adaptivity during system-level synthesis. So far, two years of activity are still missing to the completion of the project. Complete integration of the discussed methods is envisioned, as well as the precise assessment of the approach on complex and industrially-relevant case studies.

Acknowledgments

A large number of people are responsible for, or have contributed to, the work described in this paper. More specifically, we would like to thank Prof. Todor Stefanov (Universiteit Leiden), Roberta Piscitelli (Universiteit van Amsterdam), Prof. Luigi Raffo, Sebastiano Pomata and Giuseppe Tuveri (Università degli Studi di Cagliari), Prof. Peter Marwedel (Informatik Centrum Dortmund), Prof. Mariagiovanna Sami, Umberto Bondi and Onur Derin (Università della Svizzera Italiana). The research leading to these results has received funding from the European Community's Seventh Framework Programme (FP7/2007-2013) under grant agreement no. 248424, MADNESS Project, and by the Region of Sardinia, Young Researchers Grant, PO Sardegna FSE 2007-2013, L.R.7/2007 "Promotion of the scientic research and technological innovation in Sardinia".

References

[1] P. Marwedel, *Embedded System Design*. Kluwer/Springer, 2003.

[2] M. Thompson, T. Stefanov, H. Nikolov, A. D. Pimentel, C. Erbas, S. Polstra, and E. F. Deprettere, "A framework for rapid system-level exploration, synthesis, and programming of multimedia MP-SoCs," in *Proc. of the Int. Conference on Hardware-Software Codesign and System Synthesis (CODES+ISSS '07)*, 2007, pp. 9–14.

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

[4] H. Nikolov, T. Stefanov, and E. Deprettere, "Systematic and Automated Multiprocessor System Design, Programming, and Implementation," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 27, no. 3, pp. 542–555, 2008.

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

[6] *IEEE Standard for IP-XACT, Standard Structure for Packaging, Integrating, and Reusing IP within Tool Flows*, IEEE Computer Society and IEEE Standards Association Corporate Advisory Group, February 2010.

[7] S. V. Gheorghita, M. Palkovic, J. Hamers, A. Vandecappelle, S. Mamagkakis, T. Basten, L. Eeckhout, H. Corporaal, F. Catthoor, F. Vandeputte, and K. D. Bosschere, "System-scenario-based design of dynamic embedded systems," *ACM Transactions on Design Automation of Electronic Systems*, vol. 14, no. 1, pp. 1–45, 2009.

[8] P. van Stralen and A. D. Pimentel, "Scenario-based design space exploration of MPSoCs," in *Proc. of the IEEE International Conference on Computer Design (ICCD '10)*, Oct. 2010.

[9] P. Meloni, S. Secchi, and L. Raffo, "An fpga-based framework for technology-aware prototyping of multicore embedded architectures," *Embedded Systems Letters, IEEE*, vol. 2, no. 1, pp. 5 –9, 2010.

[10] P. Meloni, I. Loi, F. Angiolini, S. M. Carta, M. Barbaro, L. Raffo, and L. Benini, "Area and power modeling for networks-on-chip with layout awareness," *VLSI DESIGN*, vol. 2007, 2007.

[11] V. Nollet, D. Verkest, and H. Corporaal, "A safari through the mpsoc run-time management jungle," *Signal Processing Systems*, vol. 60, no. 2, pp. 251–268, 2010.

[12] G. De Micheli and L. Benini, *Networks on Chips: Technology and Tools*. Morgan Kaufmann, 2006.

[13] F. Angiolini, P. Meloni, S. M. Carta, L. Raffo, and L. Benini, "A layout-aware analysis of networks-on-chip and traditional interconnects for mpsocs," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 26, no. 3, pp. 421 –434, march 2007.

[14] S. Verdoolaege, *Handbook on signal processing systems*. Springer, 2010, ch. Polyhedral process networks.

[15] D. Nadezhkin, S. Meijer, T. Stefanov, and E. Deprettere, "Realizing FIFO Communication When Mapping Kahn Process Networks onto the Cell," in *Proceedings of the 9th International Workshop on Embedded Computer Systems: Architectures, Modeling, and Simulation*, ser. SAMOS '09. Berlin, Heidelberg: Springer-Verlag, 2009, pp. 308–317.

[16] M. Psarakis, D. Gizopoulos, M. Hatzimihail, A. Paschalis, A. Raghunathan, and S. Ravi, "Systematic software-based self-test for pipelined processors," in *Proceedings of the 43rd annual Design Automation Conference*, ser. DAC '06. New York, NY, USA: ACM, 2006, pp. 393–398.

[17] O. Derin, E. Diken, and L. Fiorin, "Middleware approach to achieving fault tolerance of kahn process networks on networks on chips," *International Journal of Reconfigurable Computing*, 2011.

[18] L. Fiorin, L. Micconi, and M. Sami, "Design of fault tolerant network interfaces for nocs," in *Digital System Design Architectures, Methods and Tools, 2011. DSD 2011. 14th Euromicro Conference on*, aug. 2011.

[19] O. Derin, D. Kabakci, and L. Fiorin, "Online task remapping strategies for fault-tolerant network-on-chip multiprocessors monitoring system for nocs," in *Proceedings of the 2011 Fifth ACM/IEEE International Symposium on Networks-on-Chip*, ser. NOCS '11, 2011.