Island-Based Adaptable Embedded System Design

Ivan Beretta, Vincenzo Rana, David Atienza, *Member, IEEE*, and Donatella Sciuto, *Fellow, IEEE*

***Abstract*—Nowadays, hardware devices are meant to host the execution of many complex, multicore applications, whose functional and nonfunctional requirements vary according to the specific working domain. In this work, we propose a design methodology that combines an efficient reconfigurable architecture and a related mapping flow. In particular, the proposed *island-based* hardware architecture couples an efficient area usage and an adaptable communication infrastructure. The proposed mapping flow distributes the cores on the device to optimize both performance and reconfiguration related metrics.**

***Index Terms*—CAD tools, field-programmable gate arrays (FPGAs), platform-based design, reconfigurable computing.**

I. Introduction

MODERN Systems-on-Chip (SoCs) are often required to execute different multicore applications, which are generally very complex and whose characteristics may differ according to the working scenario. These SoCs have tight functional requirements in terms of performance (which has been proved to be mainly affected by the communication among the cores [1]), area and power consumption, as well as nonfunctional requirements such as flexibility and time-to-market.

Depending on the application field, the system requirements are weighted in different ways: in real-time systems, for instance, a low switching time between the different applications is crucial to meet the strict deadlines. Conversely, in other kinds of specialized systems the switching time becomes less critical, as even a temporary interruption of the system may be acceptable. In the former case, static or parametric systems [2], [3] are generally employed, as they trade a low switching time for larger area requirements and low flexibility, i.e., no application can be added to the system at a later time. In the latter case, the system can provide optimized performance for each application by coupling traditional processing elements and reconfigurable devices [4], which allows each application to be designed independently and loaded on the device by means of a complete reconfiguration. This approach allows new applications to be added after the deployment, but it requires a high switching time (in the order of seconds [5]).

There exist other classes of problems, such as multimedia systems or wireless sensor networks (WSNs), where a low switching time is desirable to avoid a perceptible interruption of the service. On the other hand, these systems require high flexibility to remain up-to-date (e.g., to add new coding standards), while guaranteeing high performance for each application. Area requirements may also be strict, in particular in the WSNs domain where each sensor node has a limited amount of resources. Thus, it is not reasonable to deploy this kind of systems by switching among applications using a complete reconfiguration of the device, because the latency and the power consumption would be unacceptable. This can be avoided when the applications belong to the same domain, and thus they are very likely to share cores to perform common operations. In this work, we focus on this scenario, and we aim at limiting the reconfigurations to a small portion of the device by means of partial dynamic reconfiguration [6], [7], taking advantage of the similarity among the applications.

Unlike the static systems and the ones based on complete reconfiguration, the design of partially-reconfigurable SoCs is mainly carried out manually on custom hardware architectures, thus affecting the time-to-market for complex systems. Even the most modern and comprehensive methodologies, such as the one in [8], generally do not provide a framework to exploit application similarity to reduce the reconfiguration latency. To overcome these limitations, we introduce a reconfigurable architecture to provide an efficient support for applications switching, achieving an improvement of 69.6% in terms of wasted area with respect to other state-of-art architectures. Furthermore, we define an adaptive mapping flow that distributes the cores on the reconfigurable resources both at design-time and at run-time, while optimizing performance and reducing the reconfiguration latency (a 74.5% reduction is achieved with respect to a complete reconfiguration of the device).

II. Island-Based Hardware Architecture

Over the last years, networks-on-chip (NoCs) [9] have emerged as the leading technology to provide efficient and flexible communication among a large number of cores [10]. However, NoCs are generally proposed either for static systems [11], [12], or for hardware architectures based on complete reconfiguration. In the former case, the topology is designed to optimize the bandwidth of a predefined set of cores with well-known communication requirements, but new cores cannot be added at run-time. In the latter case, a separate NoC is designed for each application, though a considerable latency is introduced when it is reconfigured along with the cores.

Other works aim at combining NoCs and partial dynamic reconfiguration, such as the ones in [13], which focuses on the

Manuscript received July 20, 2010; revised October 20, 2010; accepted February 09, 2011. Date of publication February 17, 2011; date of current version June 22, 2011. This research was partially supported by the HiPEAC Network of Excellence and the Swiss National Science Foundation (SNF), under Grant 200021-127282. This manuscript was recommended for publication by R. Kastner.

I. Beretta and D. Atienza are with the Embedded Systems Laboratory, Ecole Polytechnique Fédérale de Lausanne (EPFL), Lausanne, 1015, Switzerland (e-mail: ivan.beretta@epfl.ch; david.atienza@epfl.ch).

V. Rana and D. Sciuto are with the Dipartimento di Elettronica e Informazione, Politecnico di Milano, Milano, 20133, Italy (e-mail: rana@elet.polimi.it.; sciuto@elet.polimi.it).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/LES.2011.2115991

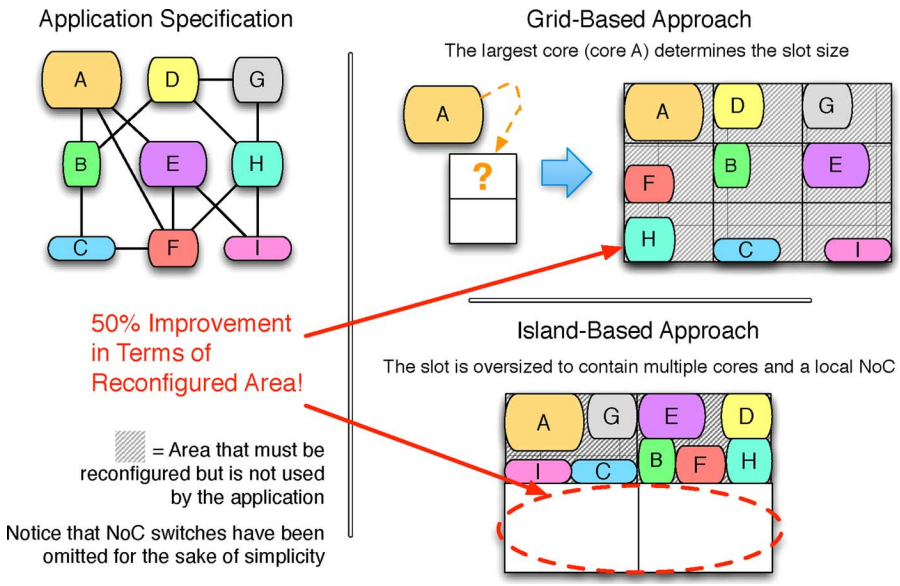

Fig. 1. Island-based hardware architecture.

hardware architecture, and in [14], where a complete design flow is proposed. In general, the idea is to design a *grid-based* architecture, where the device area is divided into small reconfigurable regions called *slots* of the same size, containing either a single core or a network switch. As each slot can be reconfigured independently, shared cores can be kept configured on the device during an application switching, thus reducing the timing overhead. The slots are connected using a NoC whose topology can be modified at run-time by adding or removing switches from the device. However, this architecture shows some structural limitations, as the number of topologies that can be implemented by placing cores and switches on the same grid is limited, and furthermore the reconfiguration of a switch makes it unavailable for a certain time, which may affect the traffic over the NoC. Finally, the slot area must be as large as the maximum core size in the application set, which may lead to a lot of wasted area, i.e., unused area that still needs to be reconfigured, as shown in Fig. 1. This is especially true when cores have different sizes, which is a scenario that frequently occurs in practice.

In this work, we propose a novel architecture which overcomes the limitations of the *grid-based* approach, and a mapping flow that efficiently takes advantage of this architecture. The proposed approach is named *island-based* architecture, and it is structured as a grid of larger reconfigurable slots that can contain one or more cores (i.e., an *island* of cores). This choice is motivated by the improved area usage, as shown in Fig. 1, because smaller cores can now be grouped in the same slot and reconfigured at once. The designer can define the number of slots, and consequently their size, according to the characteristics of the target device, and to the cores size.

We provide the *island-based* architecture with a multilevel communication infrastructure to balance flexibility and performance. A reconfigurable and local *intraslot* NoC guarantees a fast communication among the cores in the same slot: as it is reconfigured along with the cores, the local NoC is specifically designed to optimize the communication inside the slot, and it guarantees good performance while enhancing the flexibility of the system. On the other hand, a static and global *interslot* NoC guarantees connectivity among the slots, and it is never reconfigured during the application switching process. Though its topology is fixed, this static backbone allows communication to be delivered even while one or more slots are being reconfigured, thus avoiding any denial of service.

III. The Proposed Mapping Flow

The benefits of the proposed architecture can be fully exploited by trying to map cores that frequently communicate with each other into the same slot, or at least to slots that are closely connected by the intercore NoC topology. This assignment should also aim at minimizing the number of slots that are reconfigured when an application switching occurs. We propose a combined design-time and run-time mapping flow (see Fig. 2) that keeps both communication and reconfiguration related issues into account.

A. Design-Time Mapper

In the first phase of the proposed flow, a design-time algorithm maps a known set of applications before the system deployment according to the computational steps shown in Fig. 2. At first, the algorithm identifies a set of cores that should be initially deployed on the device because they are frequently used by a large number of applications. Then, the remaining cores are mapped into specific islands that are used to load the cores that are peculiar of each application [15].

The algorithm starts with the *preprocessing* stage, where the cores are sorted (*ordering* phase) according to their occurrences in the application set and their size, as larger cores are more difficult to map in fixed-sized slots and thus they should be handled first. Then, the cores achieving the highest scores are selected (*selection* phase) until the device area availability is reached, and they are used to form a graph-based data structure. In the

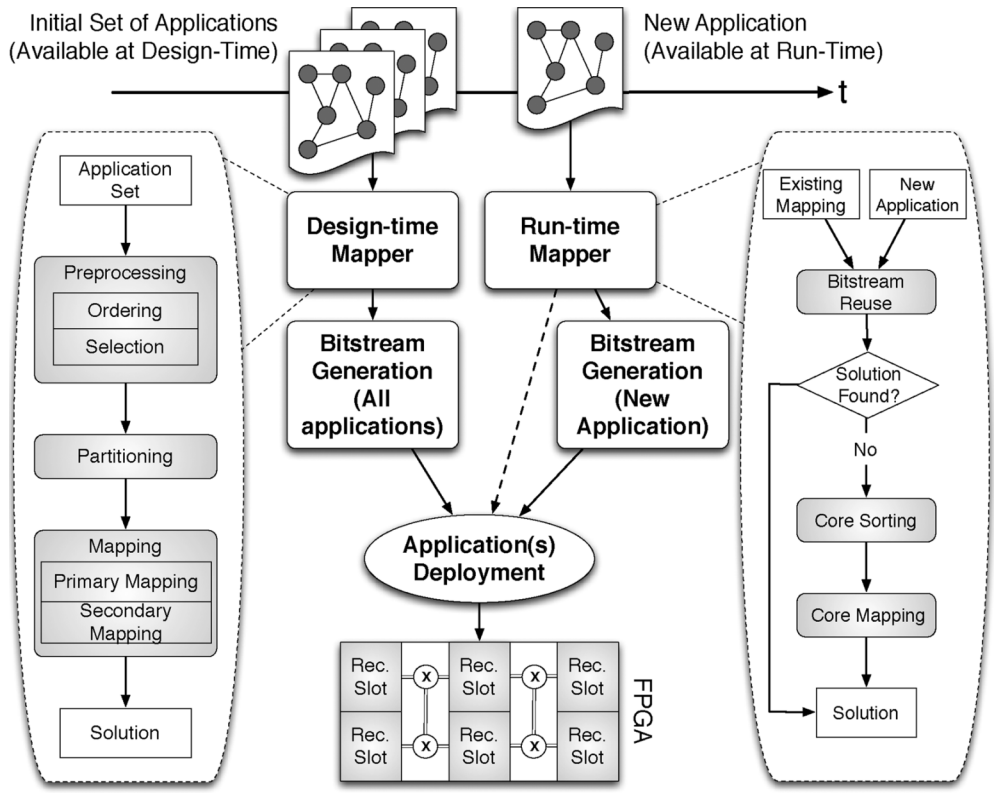

Fig. 2. Proposed mapping flow.

graph, each pair of cores is connected by an edge whose weight is proportional to the number of times the two cores appear together in the same application, which allows the algorithm to detect similarities among the applications. The graph is then processed by a partitioner (*partitioning* stage) that identifies a number of partitions equal to the number of slots, and thus generates an island for each slot in the architecture. As the partitioner aims at minimizing the weight of the edges that cross the partition boundaries, the cores connected by high-weighted edges (and thus, frequently used in the same applications) are generally clustered in the same island. Each island generated in the partitioning stage is then assigned to a specific slot on the device (*primary mapping* phase) by means of a genetic algorithm [15] that aims at reducing the traffic over the global NoC. Once the primary mapping has been completed, a binary file named *bitstream* is generated to program the entire FPGA at the system start-up.

The cores that have not been picked in the *selection* phase are peculiar of a few applications, thus they should be reconfigured on-demand when an application requires them. For each unmapped core, the algorithm identifies the best target slot (*secondary mapping* phase) by evaluating the communication between the core and the already-mapped ones. As this phase creates a set of specific islands for each application, the corresponding partial bitstreams must be generated to reconfigure the device and to switch between two applications.

B. Run-Time Mapper

In the second part of the proposed flow, we introduce run-time adaptability by defining a mapper (shown in the right-hand side of Fig. 2) that allows the dynamic addition of new applications to an existing system. These new applications may be either updates of the existing ones or brand new functionalities, and the mapper aims at reducing their deployment time. Thus, the algorithm does not modify the mapping computed at design-time, but it only focuses on the new application to detect the similarities with the already-mapped ones [16].

The deployment time of the new application is mainly affected by the bitstream generation process, thus the goal is to reduce the number of bitstreams that must be created from scratch. If some of the cores are already used by other applications, it is possible to reuse existing bitstreams (*bistream reuse* phase) to deploy part of the application, thus a new generation is not required. The algorithm selects a set of existing configurations that contain a large number of cores used by the new application, and stops when the area availability of the device becomes low. Then, the unmapped cores are ordered (*core sorting* phase) and assigned to a slot (*core mapping* phase) according to the amount of communication they generate with the already-mapped cores, also trying to use a low number of slots to keep the solution compact.

IV. Multimedia Encoding/Decoding Case Study

We analyze a real-world multimedia case study based on the encoding and decoding of a multisource video stream, in order to validate the proposed adaptable architecture and mapping flow. The standards included in the case study are *MPEG-2*, *H.263*, and *MPEG-4* [17]. On average, these applications share from 70% to 80% of their cores to perform common signal processing operations. The right-hand side of Fig. 3 shows which

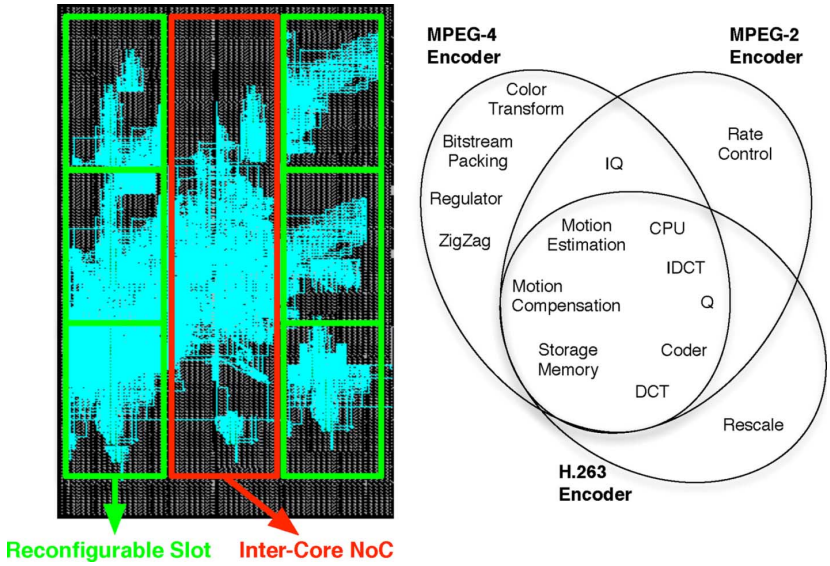

Fig. 3. Implementation of an island-based architecture on a Virtex-4 device (left), and a Venn diagram of the proposed multimedia encoders (right).

cores are required by each multimedia encoder, and which ones are in common.

A. Application Mapping

It is not possible to deploy all the proposed applications at the same time using a static system because of their large area requirements and power-related issues. Furthermore, as codecs evolve over time, a static solution would not allow the addition of a new standard after the system deployment. Nevertheless, an approach based on a complete reconfiguration of the device is also unsuitable, mainly because it would introduce a perceivable delay (in the order of seconds) every time the encoding of the input stream changes. Our approach, on the other hand, allows to configure the right decoding algorithm in a short time, in the order of few hundreds of milliseconds. Thus, we map the multimedia applications on our *island-based* architecture with six slots, which are connected using an interslot NoC with mesh grid topology, as the one shown in the left-hand side of Fig. 3 (implemented on a Xilinx Virtex-4 XC4VLX60 device [18]).

The resulting mapping deploys most of the shared cores at the system start-up, including a general-purpose CPU, a *frame buffer*, and a *motion estimation* and *compensation* units. Then, a set of specific bitstreams is generated to load the application-specific cores. For instance, two bitstreams are generated to deploy the MPEG-4 decoder: one of them includes the IDCT unit, and the other includes the *Huffman decoder*, the *inverse quantizer* and the *ZigZag* unit [17]. The IDCT is assigned to a slot that initially contains the *motion estimation* unit, while the second bitstream overwrites a slot containing the *rescale* unit, as both cores are not required during the decoding process.

B. Comparison Among Design-Time Deployment Strategies

On the selected target device, a complete reconfiguration requires a switching time of 1488 *ms*, whereas the proposed system can switch between two applications by reconfiguring an average of 1.53 slots, which requires only 380 *ms*, a 74.5% reduction. Conversely, a static or parametric system [2], [3] would not introduce any switching time because all the applications are concurrently implemented on the device, but its area requirements are up to 76% higher.

The case study also proves the benefits of the *island-based* approach with respect to the *grid-based* one [14]. We set the slot size of the *grid-based* architecture to be exactly equal to the largest core in the system (i.e., the motion compensation core): in this case, an average of 58.7% of the slot area is unused when smaller cores are mapped, which does not even allow the solution to be deployed on the selected FPGA. Furthermore, 93% of the slot area is left unused when the slot includes one of the NoC switches. Conversely, the average amount of wasted area in the *island-based* architecture is 22.7%, a 69.6% reduction with respect to the *grid-based* one, while preserving the same communication performance.

C. Run-Time System Adaptation

Now, let us assume that only the two older codecs (MPEG-2 and H.263) are available at design-time, while the MPEG-4 is added after the product release, thus the system has to be adapted to support the new standard. The initial design includes MPEG-2 and H.263 and, given their high similarity, the mapping only requires an average switching time of 32 *ms* (a 97.8% reduction with respect to a complete reconfiguration). However, 62% of the cores required by the MPEG-4 standard are not included in the original design, and the corresponding switching time requires 480 *ms*. Still, this behavior is acceptable, as it does not require any modification to the mapping of both MPEG-2 and H.263, and the MPEG-4 application can be deployed without a complete reconfiguration, which requires 1488 *ms* (thus, our run-time adaptive approach provides a 67.7% reduction in terms of reconfiguration time).

Finally, it is important to notice that the proposed mapping flow fits the time-to-market requirements of this kind of systems. In fact, the design-time mapper can compute the solution in less than one minute on a standard workstation, and it is generally followed by a bitstream generation phase that can take a

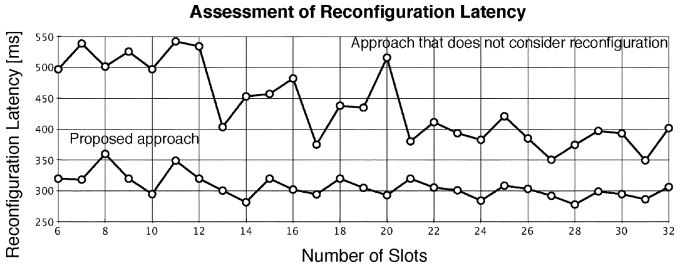

Fig. 4. Reconfiguration latency as a function of the number of slots.

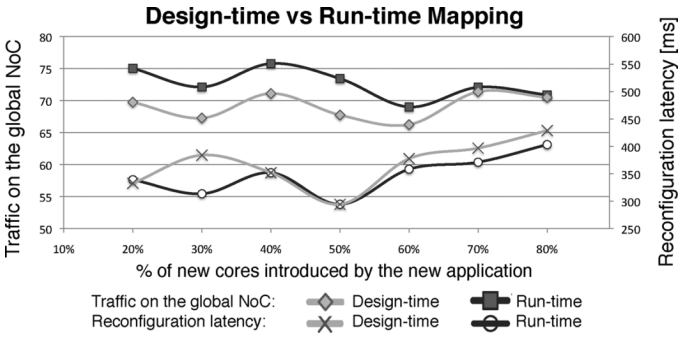

Fig. 5. Results of the run-time mapper w.r.t. the design-time approach.

few hours. The run-time algorithm can find a mapping in less than one second, and it may not even require the generation of any new bitstream.

V. Scalability Analysis

In this section, we aim at estimating the benefits of the *island-based* architecture and the related mapping flow on a set of large (i.e., 25 to 35 cores) synthetic benchmarks. The proposed applications represent typical highly-distributed multicore applications that can spread over the next years.

At first, we target a hardware architecture that includes up to 16 slots, and we aim at mapping 8 applications at design-time using the proposed flow. Fig. 4 shows a comparison between the proposed approach and a communication-oriented mapping technique which does not explicitly consider the cost of dynamic reconfiguration [15]. The results show that our mapping flow reduces the switching time by 29.1% on average, while achieving the same communication performance. Specifically, the improvement reaches 43.2% when the number of slots is low, and core grouping can be applied more effectively.

Finally, we performed additional experiments to validate the addition of new applications to an existing system. In particular, after six applications have been mapped at design-time and the system has been deployed, a seventh application becomes available at run-time. We map the new application using our run-time mapper, and we compare the results to an execution of the design-time mapper over all the seven applications, which is expected to extract similarities in a more efficient way. The results of this comparison, in terms of reconfiguration latency and an index of the amount of traffic over the global NoC, are shown in Fig. 5 as a function of the number of cores that are used only by the new application. Though the design-time algorithm outperforms the run-time mapper in terms of communication, the difference is limited to less than 5% on average, thus proving that the system is adaptable at run-time with only a minor loss of performance.

VI. Conclusion

We have proposed an MPSoC design approach which provides high flexibility as well as good performance and area usage, and a mapping flow to assign the application cores to a specific island, aiming at optimizing both communication and the switching time among different applications. A case study based on real-world multimedia applications validated the proposed approach, as the amount of wasted area is reduced by 69.6% with respect to other state-of-art architectures, and the reconfiguration latency is reduced by 74.5% with respect to a complete reconfiguration of the device. Additional experiments on synthetic benchmarks show that the proposed approach scales well on larger applications and on larger devices.

References

[1] H. G. Lee *et al.*, "On-chip communication architecture exploration: A quantitative evaluation of point-to-point, bus, and network-on-chip approaches," *ACM Trans. Design Autom. Electron. Syst.*, vol. 12, no. 3, p. 23, 2007.

[2] S. Neelkrishnan *et al.*, "Design and implementation of a parameterized noc router and its application to build prdt-based nocs," in *Proc. 5th Int. Conf. Inform. Technol.: New Generations (ITNG '08)*, Las Vegas, NV, Apr. 2008, pp. 259–264.

[3] M. Dall'Osso *et al.*, "Xpipes: A latency insensitive parameterized network-on-chip architecture for multiprocessor socs," in *Proc. 21st Int. Conf. Comput. Design (ICCD '03)*, San Jose, CA, Oct. 2003, pp. 536–539.

[4] E. Flamand, "Strategic directions towards multicore application specific computing," in *Proc. Design Autom. Test Eur. Conf. Exhib. (DATE '09)*, Nice, France, Apr. 2009, p. 1266.

[5] V. Rana *et al.*, "A reconfigurable network-on-chip architecture for optimal multi-processor SoC communication," in *Proc. Design Methodologies for SoC and SiP*, 2009.

[6] Early Access Partial Reconfiguration User Guide Xilinx, Mar. 2006.

[7] C. Kao, "Benefits of partial reconfiguration," *XCell J.*, pp. 65–67, 2005.

[8] K. Schleupen *et al.*, "Dynamic partial FPGA reconfiguration in a prototype microprocessor system," in *Proc. Field Program. Logic Appl. (FPL)*, Amsterdam, The Netherlands, 2007, pp. 533–536.

[9] L. Benini and G. De Micheli, "Networks on chips: A new soc paradigm," *Computer*, vol. 35, no. 1, pp. 70–78, Jan. 2002.

[10] E. Bolotin *et al.*, "Cost considerations in network on chip," *Integr. VLSI J.*, vol. 38, no. 1, pp. 19–42, 2004.

[11] A. Hansson and K. Goossens, "Trade-offs in the configuration of a noc for multiple use-cases," in *Proc. 1st Int. Symp. Netw.-on-Chip (NOCS)*, Princeton, NJ, May 2007, pp. 233–242.

[12] L. Benini, "Application specific noc design," in *Proc. Autom. Test Eur. Conf. Exhib. (DATE '06)*, Munich, Germany, 2006, vol. 1, pp. 1–5.

[13] T. Pionteck *et al.*, "Applying partial reconfiguration to networks-on-chips," in *Proc. Field Program. Logic Appl. (FPL)*, Madrid, Spain, Aug. 2006, pp. 1–6.

[14] D. Cozzi *et al.*, "Reconfigurable noc design flow for multiple applications run-time mapping on fpga devices," in *Proc. 19th ACM Great Lakes Symp. VLSI (GLSVLSI '09)*, Boston, MA, 2009, pp. 421–424.

[15] V. Rana *et al.*, "Minimization of the reconfiguration latency for the mapping of applications on FPGA-based systems," in *Proc. Int. Conf. Hardware/Software Codesign Syst. Synth. (CODES-ISSS '09)*, Grenoble, France, 2009, pp. 325–334.

[16] I. Beretta *et al.*, "Run-time mapping of applications on fpga-based reconfigurable systems," in *Proc. 19th ACM Great Lakes Symp. VLSI (GLSVLSI '09)*, Boston, MA, 2010, pp. 421–424.

[17] J. Niu *et al.*, "Mpeg-4 video encoder based on dsp-fpga techniques," in *Proc. Int. Conf. Commun. Circuit., Syst. (ICICS '05)*, Hong Kong, 27–30, 2005, vol. 1, pp. 518–522.

[18] Virtex-4 FPGA User Guide (UG070) Xilinx, 2008.