Self-Adaptive Networked Entities for Building Pervasive Computing Architectures

Martin Danek*, Jean-Marc Philippe°, Roman Bartosinski*, Petr Honzik*, and Christian Gamrat° ⋆

*Department of Signal Processing, UTIA AV CR
Pod Vodarenskou vezi 4
Praha 8, 182 08, Czech Republic
danek@utia.cas.cz

°CEA, LIST,
Boîte Courrier 65,
Gif-sur-Yvette, F-91191 France;
jean-marc.philippe@cea.fr

Abstract. This paper presents a framework for building and modeling a new-generation self-adaptive systems. The first part of the paper proposes an architecture of a self-adaptive networked entity that forms the basic element of the approach. The second part describes a modeling environment based on Matlab / Simulink and one possible implementation of the self-adaptive networked entity. A physical realization of the proposed system is demonstrated on the computation of a simple FIR filter in several FPGAs acting as hardware in the loop in Matlab / Simulink.

1 Introduction

In the past years, we have observed a gradual shift in the use of high performance computing resources. After the mainframe era (many people, one computer) and the PC era (one computer per person), we begin to enter the ubiquitous or pervasive computing era (many computers per person) [1]. Computer related devices are now a critical part in almost any piece of modern equipment. Every day an average citizen interacts with at least one electronic device equipped with a processing element. Some of these devices are already even capable of communicating with others, for example a cell phone or personal computer. This shift is being accelerated by progress in micro/nano-electronics technology allowing for smaller and more power-aware chips.

An important factor that must also be considered is that in the 65nm technology fabs can produce with a reasonable yield only regular patterns such as FPGA circuits or networked CPU arrays. If this trend is not changed, we will

⋆ This work was supported and funded by the European Commission under Project ÆTHER No. FP6-2004-IST-4-027611, and by the Czech Ministry of Education under Project CAK No. 1M0567

be faced with a more intense need for design tools and methods that can divide and implement computation in multiple distributed targets with different granularity. This increase in the number of processors is true at all levels of the computing systems, from chips (multi-cores) to system level. Moreover, all these intelligent devices will be interconnected through an intricate chain of networks.

This paper deals with the hardware part of the ÆTHER project [2] and introduces a novel architecture based on several automomous computing units that we call **Self-Adaptive Networked Entities (SANEs)**. As its name indicates, this basic computing element aims at being networked with others in order to form complex but manageable systems. The remainder of the paper is as follows. The second section deals with a short state of the art based on relevant project in the field. The third section introduces and describes the architecture. Then the fourth section presents some possible implementations of the SANE hardware followed by the description of the modelling. The sixth section present current implementation with accent to self-adaptive network and easy modeling. Finally, the seventh section concludes the paper.

2 State of the Art

On the international scientific scene, some other projects are scientifically linked with the ÆTHER proposal at the hardware level. In the U.S., the High Productivity Computing Systems (HPCS) program [3] of the U.S. DARPA agency aims at developing a broad spectrum of innovative computing system solutions that will fill the gap between today's HPC computing architectures and the promise of future quantum computing. The Self-Regenerative System program [4] aims at bringing more autonomy to computing systems by introducing such capabilities as self-optimisation, self-diagnosis, self-awareness and self-healing. The MIT Oxygen project [5] gathers academic labs and industrials around the topic of adaptable, efficient and powerful computing resources for pervasive applications. They can delegate part of their computation in order to conserve power and they can perform a wide variety of functions thanks to reconfiguration.

In Japan, the T-Engine forum [6] aims at developing an open, standardised, real-time platform for future ubiquitous systems. The project was funded by five Japanese chipmakers and 17 other technological firms. From now on, the consortium is much more important (443 members as of February 29th, 2008). The underlying hardware is composed of four dedicated boards [1].

On the European side, several projects in the pervasive computing domain are under way. The RUNES project [7] aims at enabling the creation of large-scale, widely distributed, heterogeneous networked embedded systems that interoperate and adapt to their environments. The PalCom project [8] aims at researching and developing a new perspective on ambient computing denoted palpable computing.

3 The Self-Adaptive Networked Entity

We define self-adaptivity as the ability of a system to adapt to an environment by allowing its components to monitor this environment and change their behaviour in order to preserve or improve the operation of the system according to some defined criteria [2]. The environment of a system is defined by everything that interacts with this system. The adaptation of a system can occur at different levels, from hardware to software. In this article, we study self-adaptivity at the hardware level and especially with reconfigurable architectures.

Some requirements of the underlying hardware can be enumerated from the definition. Since the architecture must adapt, the underlying hardware has to be reconfigurable in order to allow a deep change in the hardware structure. The architecture also needs to efficiently monitor its environment and its behaviour in order to be aware of its performance and the possible related improvements. It also needs a decision taking mechanism to decide the moment and the nature of the adaptation. The combination of the monitoring process and the decision taking process provides the device with the ability to self-adapt: it can automously trigger reconfigurations to improve or to keep its performance after a modification in its environment.

3.1 Description of the SANE

The proposed architecture is based on a set of elementary computing entities named Self-Adaptive Networked Entities (SANE) partly presented in [9]. The SANE is defined as a computing entity at which a local and autonomous decision process occurs that affects its own operation. Figure 1 shows a functional view of the SANE.

The computing engine processes data. It must provide the necessary processing power to handle future complex algorithms required by new standards and multimedia applications. It must also be reconfigurable to handle a wide spectrum of applications, and the reconfiguration process must be dynamic (”on the fly” reconfiguration). It is the most flexible block of the SANE.

The observer is responsible for monitoring the computation process and other runtime parameters. It allows the SANE to sense its current environment and optimize its performance, which means both comparing its actual processing parameters with the required constraints given by the application designer, and monitoring communication parameters. As an example optimal working mode can be driven by the bit-error rate measurement on a link in order to automatically set the correct power supply voltage and frequency [10] or the appropriate error detection code [11].

The controller is in charge of actually taking all decisions regarding the ongoing computation task. Based on the observations of the current running task, the controller is responsible for changing the state of the SANE by loading any locally available task implementation.

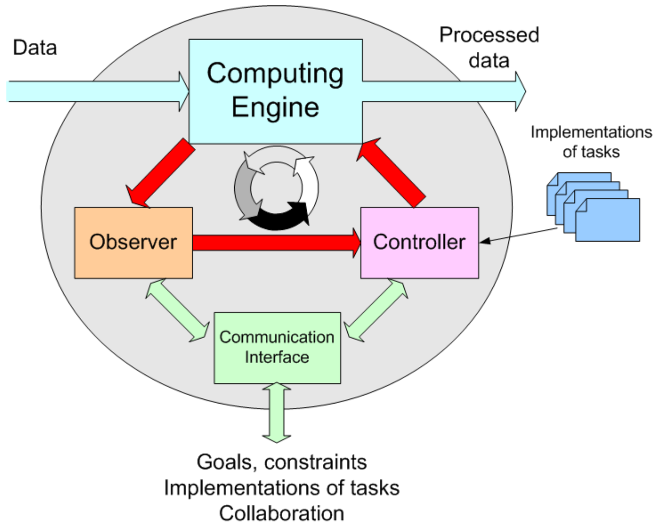

Fig. 1. Functional view of the SANE with the main parts.

The communication interface is responsible for the management of SANE assemblies. It enables resources sharing and collaboration between SANE elements as well as providing the SANE with goals to be reached.

The SANE is basically a tightly coupled hardware / software entity with its computing engine seen as a hardware-based reconfigurable computing unit (e.g. an FPGA fabric), and the observer and the controller implemented as more software programmable CPU cores.

3.2 Self-Adaptive Network

The previous text describes the SANE as a self-adaptive networked entity embedded in various ways such as pervasive computer around us. The SANE can be seen as a computer for their main purpose, or it can adapt itself for current require function and attach itself to virtual network. Self-adaptivity becomes an advantage for the whole network built from SANEs.

Current networks need to transfer a huge amount of data between nodes that process the data flow [12]. The bandwidth and delay depend on the size of the data and distance between nodes with the demanded function [13], [14]. It is not rarely the case that data run through half of the network before they reach the demanded function. The SANE network eliminates the long distance data transport. Self-adaptation allows to change the SANE function to the required one and the resulting data can be processed by the neighboring SANE. This leads to data being processed by the closest SANE.

To reach optimal adaptation of each SANE statistical information about passed packets is collected in each SANE in the network. The passed packets carry information about prescribed processing. Each SANE knows the most required function to process the passed data after the data have moved in the network for some time. SANEs will adapt themselves to work optimally. When the prescribed processing in the packet changes the optimal function a new placement will be reached in a short time.

4 Implementation Background

This section presents implementation and simulation background necessary for the development of the SANE element that is applicable to a wide variety of current FPGAs. The efficiency of the proposed approach is evaluated in the frame of typical FPGA implementations of DSP-type computations, since the DSP domain is the primary area of today's FPGA implementations.

4.1 Dynamic Reconfiguration

Dynamic reconfiguration is a feature found in some field-programmable gate array (FPGA) families that enables to change the function of a part of the internal logic while another part continues uninterrupted in its function. This is an important feature for building self-$*$ circuits when a suitable methodology becomes available. The major contributions to the area of dynamic reconfiguration of FPGA circuits relevant to this paper are summed up in [15], [16], [17], [18], [19], [20].

The infrastructure provides support for using dynamic reconfiguration, here represented by loading and clearing a precompiled (i.e. pre placed and routed) IP core in an FPGA on the fly. The FPGA is divided into a static part and a dynamic part. The details are described in [16].

4.2 Design Flow Based on Simulink and the DK Design Suite

The proposed design flow for building systems based on SANEs is based on the bit-exact modeling of SANEs, programmed in Handel-C, in the Simulink framework, where the Handel-C code is developed in the Celoxica DK Design Suite combined simulation and synthesis environment. This enables the designer to decompose the whole system into parts with simple functions with rapid development and test of different combinations.

For the whole SANE development it is crucial to have a robust and fast design and simulation framework, usable for complex designs, to prove the equivalence between the proposed theoretical principles coded in Simulink and the final design implementation in an FPGA.

Create a SANE Model in Simulink. First we build a model of the SANE in Simulink. The data sources and sinks in this model will be the BRAMs shared with CPU in the final implementation. Since the FPGA operations are (or can be) written as a cycle-accurate and bit-exact Handel-C code, we can benefit from having a single source for both implementation and simulation by compiling the

Handel-C code to a DLL library that can be linked to the Simulink environment and called as bit-exact S-functions.

Perform Cycle Accurate Verification. Our next stage is to create test vectors using Simulink and feed these into the bit-exact and cycle-exact accurate simulation of the SANE in the DK Design Suite's debugger.

Test SANE in Hardware. We take advantage of a layered design approach by using a single communication API for data I/O functions that applies for both simulation and implementation. This allows us to verify the SANE design on real FPGA hardware by 'linking' with an appropriate board support library for implementation. We can optionally insert this hardware test back into the Simulink model for *hardware-in-the-loop* simulations.

Create a Model of a SANE Network in Simulink. We build the SANE network using a Simulink blockset (described in the next section) and the SANE model developed and debugged in the previous steps. We use the Simulink environment to emulate a dataflow network with an arbitrary topology.

Evaluate the Behaviour of the SANE Network in HW. We use the SANE network created in the previous step, but we replace the SANE models in Simulink with their hardware realization in Celoxica RC10 boards that act as hardware in the loop.

At present we use one RC10 board per one SANE to avoid the complexity of partial runtime reconfiguration, necessary when the whole SANE network is implemented in one FPGA. Each board uses static reconfiguration of its FPGA to configure different SANE functions during runtime, to decrease the amount of data circulating in the SANE network all possible SANE configurations are stored within each board in its configuration FLASH memory.

5 Modelling and Implementing a Simple SANE Network

Imagine a network of SANE entities that implement a family of FIR filters. The requirement is to share the resources among different data channels with different data throughput and filtering requirements. The task of the network is to process data streams, i.e. perform different filtering operations on the input data and generate responses. The network operates on tagged data packets. Each packet is formed by a header and data part. The header specifies in some way operations needed to process the data part; it contains the packet length, array of tags mark operations of data, the end-of-tag-array delimiter and data array length (see Figure 4).

The whole network of FIR SANE elements can be viewed as a linear chain of processing elements with a FIFO memory forming a loop to circulate data not completely processed (see Figure 2). The use of this topology (ring) enables us to concentrate on the SANE computation only, since the ring can emulate any given network topology on a logical level. We are aware of the importance of a proper networking scheme, it will be addressed in more advanced stages of this research.

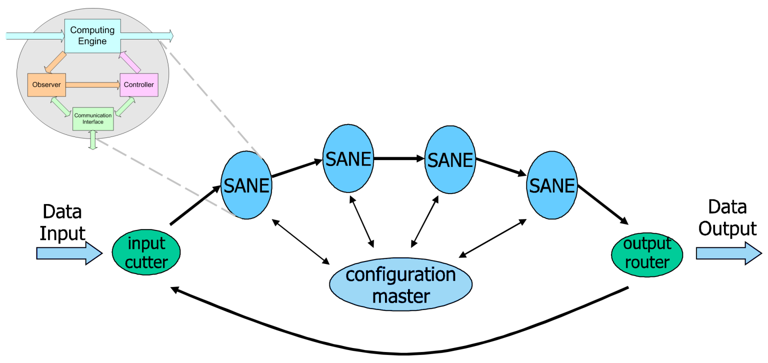

Fig. 2. General view of the SANE network.

The SANE network is modelled in the Matlab / Simulink environment and verified on the Celoxica RC10 boards acting as hardware in the loop. The idea is to use the Simulink environment for modelling, visualization and debugging. A major advantage of this approach is that it allows us to gradually move from the software model to the hardware implementation. Another advantage is the possibility to use any FPGA platform provided it supports a standard simple data exchange protocols; this way we can easily mix implementations on different boards and with FPGAs from different families and manufacturers.

5.1 SANE Model in Simulink

The model of SANE network in Simulink is based on the following four blocks that form the Simulink SANE blockset.

The input cutter takes an input data stream and divides it into packets of data. In addition the cutter prepares packet headers for the data. In the simplest case a header contains an ordered array of tags that indicate operations that are required to process the data.

The output router is responsible for directing processed data out of the network, and to direct partially processed data packets back to the network. The processed data are recognized by an empty array of tags. Partially processed data are stored in a FIFO memory in the feedback loop (not shown).

The configuration master is responsible for managing the database of configuration bitstreams and programs for the FPGA and CPU. The master sends configurations on demand from individual SANE elements. The master by no means introduces central control to the SANE network since this would invalidate the distributed processing character of the experiment.

The SANE element monitors the character of data that pass through it, it processes data packets with tags that match its functionality. The distributed control of the network is implemented as individual local decisions by each SANE to change its internal configuration to match the majority of passing data tags

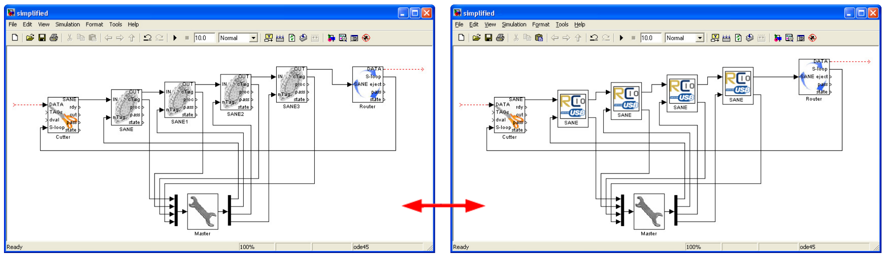

Fig. 3. SANE model, simplified view, SW model (left) and hardware in the loop (right).

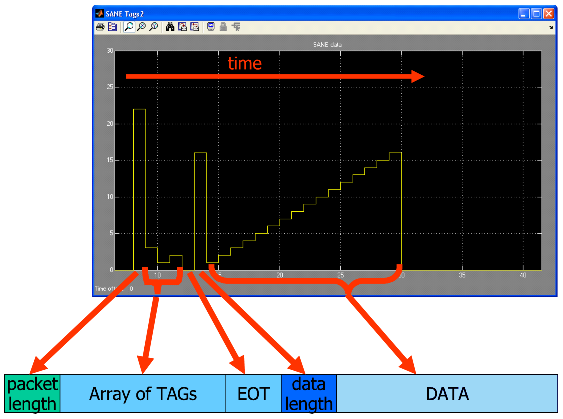

Fig. 4. SANE model, interpretation of signal plots.

We use the above building blocks to assemble a Simulink schema that models the SANE network. A clearer view is showed in Figure 3; the left part shows a situation where the whole network is modelled in Simulink, the right part shows SANEs implemented in individual FPGA development boards and connected to the Simulink environment as hardware in the loop.

Figure 4 shows how SANE packets are mapped to Simulink data. Packets are formed by consecutive numbers that are visualised as Matlab value plots.

6 Current Hardware Implementation

We use the above described FIR filter in our research to validate the proposed theory on existing hardware and to suggest requirements on a future generation of reconfigurable architectures. The system uses an external memory to store partial FPGA bitstreams. At present the complete FPGA is reconfigured statically to a new function using the built-in infrastructure of the RC10 boards; The plan is to use dynamic reconfiguration managed by a soft-core CPU that handles FPGA reconfiguration and interfaces the system to the outside world, and an FPGA portion denoted as a wrapper with a virtual socket that can be reconfigured to desired functions by the partial bitstreams (see [16]). In the described example we use reconfiguration to change the parameters and the structure of the FIR filters during runtime.

6.1 SANE Implementation

The SANE is implemented on the Celoxica RC10 board. A communication between the SANEs and IBM PC runs through the USB (main communication) and RS232 (debugging) interfaces. This arrangement allows to monitor and control the communication channel. The multi-layer solution (see Figure 5) was used to implement the SANE hardware:

The device layer services the data, control and debug channels on the SANE. It is built as a universal input-output engine. By changing the layer we can use different hardware or different NoC.

The control layer decodes packets, decides how to process them, and collects statistical information on passed and processed packets. According to this information the control layer chooses different functions for the SANE to process data packets when tags match the SANE function, or pass the packet back to the network.

The application layer processes the data. The data from a packet with a tag that matches an available SANE function are passed to the application layer. The processed data are passed back to the control layer that generates an updated packet.

Each SANE layer contains parallel function blocks, or servers, shown in Figure 5; the servers working simultaneously and communicating through signals. A received packet is stored in a Packet Buffer. Servers have access to the Packet Buffer to proccess the packet header. A packet qualified for processing in the current SANE is moved from the control layer to the application layer. During packet processing another packets can be examined in the Packet buffer.

6.2 SANE Adaptation

At the beginning the SANE is not configured to perform any function. During the run an observer chooses the best configuration for its computing engine based on the collected statistical information on tags of the packets that passed through the SANE. From the moment the SANE is configured, it can process packets with tags that correspond to its function. The SANE function can be switched according to the current statistical information on data packets. The statistical algorithm is modular, and it can be changed or debugged easily to test different aproaches to self-adaptation.

6.3 SANE Communication

The SANE implemented in the Celoxica RC10 board is connected to Matlab / Simulink. The connection is done through USB in the star topology at the physical level. The Matlab / Simulink environment provides a platform to generate

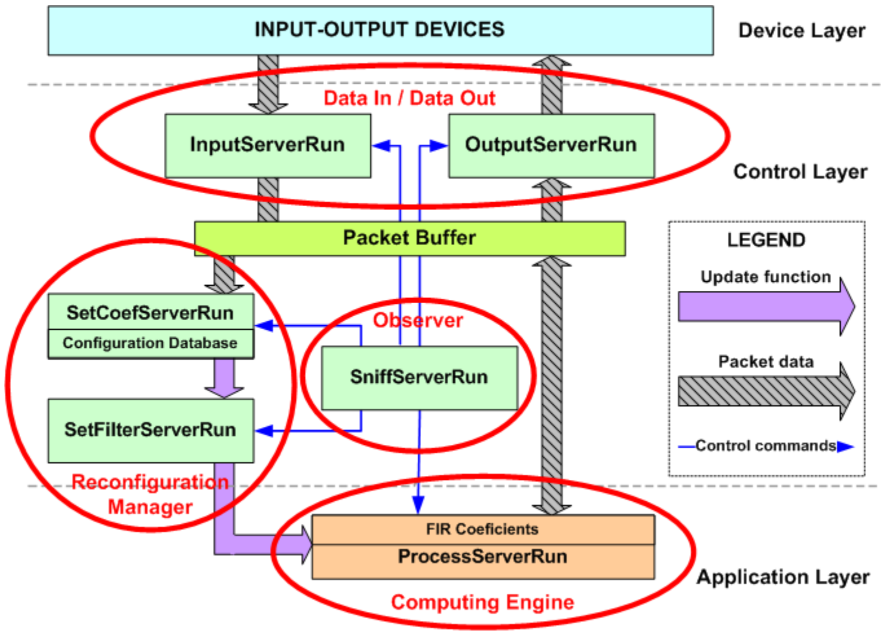

Fig. 5. SANE hardware block diagram and multi-layer solution.

and validate data. The virtual SANE network is built in Matlab / Simulink to allow the possibility to simulate various topologies such as crossbar or mesh that are often used in tile processor platforms [21]. The current SANE virtual network is built as a ring to provide high throughput and a simple implementation of the network interface [22]. The ring allows packets to cycle in the network till they have been processed completely. The ring topology allows to avoid switching and routing difficulties and focus on the development of self-adaptation and SANE debugging.

The Matlab / Simulink network model allows to connect and disconnect the SANE during operation without any network and data damage. It allows us to examine the influence of the number of SANEs in the network and to force the SANE network to change configuration. An example of the behaviour of a SANE network with four SANEs is shown in Figure 6. The network is configured to calculate a FIR filter response to a unit impulse in one data pass through the SANE network. The plots show the result when two, three or four SANEs are active. The partly processed data can be finished either in additional data passes through the SANE network, or by additional SANEs; the first case results in the computation taking more time, the second case requires more computing resources.

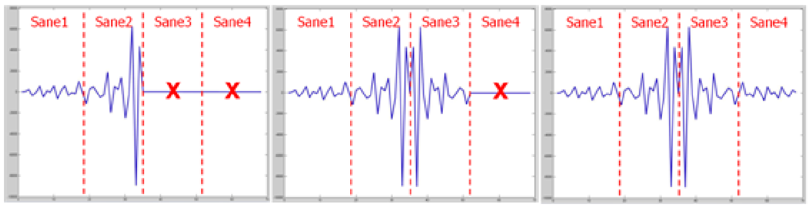

Fig. 6. FIR impulse response processed by two, three and four SANEs.

6.4 Current Implementation Advantage

The implementation of the SANE network presented above was designed to ease SANE debugging using the advantage of Matlab / Simulink and hardware in the loop simulation. Matlab provides a well known platform to generate and validate data. Simulink provides modeling interface between various hardware platforms, and it allows to change the network topology at a low cost. The bandwidth of the SANE network is not significant in the current implementation. The SANE designed on the current platform can be used as a starting point for SoC solutions as a well-debugged building block

7 Conclusions

This paper has presented a novel self-adaptive element called Self-Adaptive Networked Entity (SANE) that forms a basis for building a new class of self-adaptive systems. This autonomous element is intended to be networked with other SANE elements to form complete systems. A design flow has been presented as well as a possible model and SANE implementations using reconfigurable hardware. The presented implementation focuses on debugging the SANE design and its self-adaptation, with taking all the advantages of the Matlab / Simulink environment to interact with hardware.

References

1. J. Krikke, “T-engine: Japans ubiquitous computing architecture is ready for prime time,” *IEEE Pervasive Computing*, vol. 4, no. 2, pp. 4–9, 2005.
2. *The AETHER project web page - http://www.aether-ist.org.* The AETHER consorcium, 2006.
3. http://www.darpa.mil/ipto/programs/hpcs/.
4. http://www.darpa.mil/IPTO/Programs/srs/index.htm.
5. http://oxygen.csail.mit.edu/.
6. http://www.t engine.org/.
7. http://www.ist runes.org/.
8. http://www.ist palcom.org/.

9. K.Paulsson, M.Hübner, J.Becker, J.-M.Philippe, and C.Gamrat, “On-line routing of reconfigurable functions for future self-adaptive systems - investigations within the AETHER project,” in *Proceedings of the International Conference on Field Programmable Logic and Applications (FPL 2007)*, 2007.
10. F. Worm, P. Ienne, P. Thiran, and G. De Micheli, “An adaptive low-power transmission scheme for on-chip networks,” in *Proceedings of the ACM International Symposium on System Synthesis (ISSS'02)*, 2002.
11. L. Li, N. Vijaykrishnan, M. Kandemir, and M. J. Irwin, “Adaptive error protection for energy efficiency,” in *Proceedings of the International Conference on Computer Aided Design (ICCAD'03)*, 2003.
12. J. A. Kahle, M. N. Day, H. P. Hofstee, C. R. Johns, T. R. Maeurer, and D. Shippy, “Introduction to the Cell multiprocessor,” *IBM J. Res. Dev.*, vol. 49, no. 4/5, pp. 589–604, 2005.
13. M. Millberg, E. Nilsson, R. Thid, and A. Jantsch, “Guaranteed bandwidth using looped containers in temporally disjoint networks within the nostrum network on chip,” 2004. [Online]. Available: citeseer.ist.psu.edu/millberg04guaranteed.html
14. P. Guerrier and A. Greiner, “A generic architecture for on-chip packet switched interconnections,” 2000. [Online]. Available: citeseer.ist.psu.edu/guerrier00generic.html
15. *The RECONF2 project web page - http://reconf.org.* The RECONF2 consortium, 2002.
16. R. Bartosinski, M. Daněk, P. Honzík, and R. Matoušek, “Dynamic reconfiguration in FPGA-based SoC designs.” in *Proceedings of IEEE Design and Diagnostics of Electronic Circuits and Systems Workshop DDECS2005*, J. S. et al., Ed. University of West Hungary, 2005, pp. 129–136.
17. E. L. Horta, J. W. Lockwood, and D. Parlour, “Dynamic hardware plugins in an FPGA with partial runtime reconfiguration,” in *Proceedings of the 39th Design Automation Conference*, B. Ackland, Ed. ACM Press, 2002, pp. 343–348.
18. R. Kielblik, J. M. Moreno, A. Napieralski, G. Jablonski, and T. Szymanski, “High-level partitioning of digital systems based on dynamically reconfigurable devices.” in *Proceedings of the 12th International Conference on Field-Programmable Logic and Applications*, M. Glesner, P. Zipf, and M. Renovell, Eds. Springer, 2002, pp. 271–280.
19. M. J. Wirthlin and B. L. Hutchings, “Improving functional density using run-time circuit reconfiguration,” *IEEE Trans. VLSI Syst.*, vol. 6, no. 2, pp. 247–256, Feb. 2002.
20. I. Robertson, J. Irvine, P. Lysaght, and D. Robinson, “Timing verification of dynamically reconfigurable logic for the Xilinx Virtex FPGA series.” in *Proceedings of the 10th International Symposium on Field-Programmable Gate Arrays*, M. Schlag and S. Trimberger, Eds. ACM Press, 2002, pp. 127–132.
21. D. Wentzlaff, P. Griffin, H. Hoffmann, L. Bao, B. Edwards, C. Ramey, M. Mattina, C.-C. Miao, J. F. B. III, and A. Agarwal, “On-chip interconnection architecture of the Tile processor,” *IEEE Micro*, vol. 27, no. 5, pp. 15–31, 2007.
22. C. I. of Electrical and I. S. Electronics Engineers, *IEEE Standard for Scalable Coherent Interface. IEEE Std. 1596-1992.* New York, USA: IEEE Standards Office, 1993.