Embedded Virtualization for the Next Generation of Cluster-based MPSoCs

Alexandra Aguiar, Felipe G. de Magalhães, Fabiano Hessel
Faculty of Informatics – PUCRS – Av. Ipiranga 6681, Porto Alegre, Brazil
alexandra.aguiar@pucrs.br, felipe.magalhaes@acad.pucrs.br, fabiano.hessel@pucrs.br

Abstract

Classic MPSoCs tend to be fully implemented using a single communication approach. However, recent efforts have shown a new promising multiprocessor system-on-chip infrastructure: cluster-based or clustered MPSoC. This infrastructure adopts hybrid interconnection schemes where both buses and NoCs are used in a concomitant way. The main idea is to decrease the size and complexity of the NoC by using bus based communication systems at each local port. For example, while in a classic approach a 16 processor NoC might be formed in a 4 x 4 arrangement, in cluster-based MPSoCs a 2 x 2 NoC is employed and each router connected to a local port contains buses that carry 4 processors. Nevertheless, although good results have been reached using this approach, the implementation of wrappers to connect the local router port to the bus can be complex. Therefore, we propose in this work the use of embedded virtualization, another current promising technique, to achieve similar results to cluster based MPSoCs without the need for wrappers besides providing a decreased area usage.

1 Introduction

Embedded Systems (ES) have become a solid reality in people's lives. They are present in a broad range of facilities, such as entertainment devices (smart phones, video cameras, games toys), medical supply (dialysis machines, infusion pumps, cardiac monitors), automotive business (engine controls, security, ABS) and even in aerospace and defense fields (flight management, smart weaponry, jet engine control) [18].

Usually, these systems need powerful implementation solutions, which contemplates several processor units, such as the Multiprocessor System-on-Chips (MPSoCs) [11]. One of the most important issues regarding MPSoCs lies in the way communication is implemented. Initially, bus-based systems used to be the most common communication solution, since they used to be usually simpler in terms of implementation.

On the other hand, buses have poor scalability rates, since only a few dozens of processors can be placed in the same structure without presenting prohibitive contention rates. Therefore, other communication solutions started being researched and the most prominent one is the Network-on-Chip (NoC) approach [16].

NoCs are a communication solution widely accepted and based on general purpose network concepts. However, NoCs can present more complex communication protocols and, consequently, less predictability.

In this context, a recent idea known as Cluster-based MPSoCs has gained notoriety [7], [12]. In this approach the best of both worlds are intended to be placed together: NoCs allow higher scalability rates but buses keep the design simpler even with more processors on the system. To better understand the concept, Figure 1 depicts a 2x2 sized NoC which contains a bus located at each local port. Each bus carries along four processors which communicate in simpler ways inside and, if needed, can communicate with other clusters through the NoC. Dotted lines represent the wrappers needed to connect the bus to the NoC.

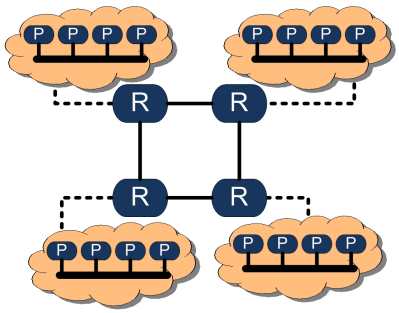

Figure 1. Cluster-based MPSoC concept

Another recent idea for embedded systems is the use of virtualization in their composition. Virtualization has several possible advantages, including the decrease of area, increase of security levels and the ease of software design [10], [2], [3]. Virtualized systems are composed by

978-1-4577-0660-8/11/$26.00 2011 IEEE

a hypervisor that holds and controls all virtual machines' operation details.

This paper proposes the unification of both concepts. Instead of using buses on each router of the NoC, we propose a single processor holding a hypervisor, providing the emulation of several virtual processors. Since buses are poorly scalable, hypervisors do not need to support more processors than a simple bus would. The main contribution of this proposal, named as Virtual Cluster-based MPSoCs, is to provide multiprocessed systems with less area occupation.

The remainder of the paper is organized as it follows. Next section show some related work on cluster-based MPSoC. Section 3 shows basic concepts regarding embedded virtualization. Then, in Section 4, details about the Virtual Cluster-based MPSoCs are discussed. Section 5 details motivational use cases and some initial experimental results. Finally, Section 6 concludes the paper besides presenting some future work.

2 Cluster-based MPSoCs

It is widely known that several MPSoCs are bus-based architectures. Systems such as the ARM MPCore [9], the Intel IXP2855 [6] and the Cell processor [13] are examples of it. Nevertheless, the need for more processing elements and a growing system complexity has led other approaches to be researched.

Networks-on-Chip (NoCs) have arisen as the main communication infrastructure involving complex MPSoCs. However, the design of NoC-based parallel application is far more complex that the one involving only bus-based systems [7].

Due to the lack of scalability present in bus-based systems and the excessive application design complexity found in NoCs, cluster-based systems are becoming a possible alternative. These systems, intend to achieve the advantages of both systems.

In [7], the authors propose a cluster-based MPSoC prototype design. In this paper, the authors integrate 17 NiosII [14] cores, organized in four processing clusters and a central core. In every cluster, the cores are composed by their own local memory and their communication is performed through a shared memory, accessed from the bus. In order to access the inter-cluster communication, cores have a shared network interface.

This system still proposes that a single processing element has the access to external peripherals, such as SDRAM controllers. Also, this central control unit is responsible for managing mapping issues of the parallel application in the clusters as well as gathering expected results. Figure 2 depicts the architecture proposed by [7]. In this Figure, LM stands for Local Memory, CSM, for Common Shared Memory, NI, for Network Interface and SDRAM IF for SDRAM Interface.

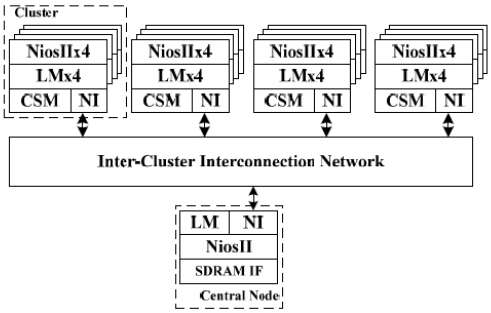

Figure 2. The architecture of Cluster-based MPSoC proposed by [7]

Results were taken considering two real applications: matrix chain multiplication and JPEG picture decoding, both implemented on an FPGA development board. The implementation resulted in speedup ratios of above 15 times. The main drawback is that real-time applications are not referred by the authors.

Figure 3 shows an example of a processing cluster that composes the cluster-based MPSoC, which is composed by four processor cores itself. Each processor core, a NIOSII, contains its own Local Memory (LM, in the figure) and a bridge to access the local bus. In this bus, it is also connected a Common Shared Memory (CSM, in the figure), used to exchange data among the processors. Still, a semaphore register file, used for synchronization purposes among the processes during the use of the shared memory, is present. Finally, the cores also share a Network Interface (NI, in the figure) which allows the inter-cluster communication.

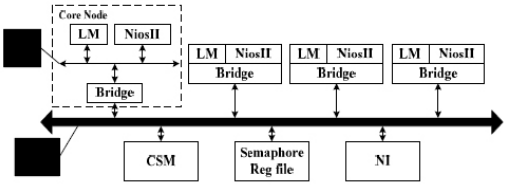

Figure 3. The architecture of each processing cluster proposed by [7]

Jin [12] proposes a cluster-based MPSoC using hierarchical buses on-chip, aiming to attack some of the problems pure NoC implementations can present to the component connected to the network. One of the main problems pointed by the authors is for real-time applications,

where the NoC must provide a high efficiency for data exchange. In this approach, no NoCs are adopted. Therefore, in cluster-based MPSoCs the performance of the computation cluster is very important for the system as a whole.

The approach presented in [12] can be seen in Figure 4. The system adopts the AMBA-AHB protocol, which is a high performance system bus that supports multiple bus masters besides providing high-bandwidth operation. The authors also use a hierarchical bus architecture aiming to obtain better performance results, especially when decreasing bus collision rates, improving the speed of register configuration and avoiding shared memory contention and bottlenecks.

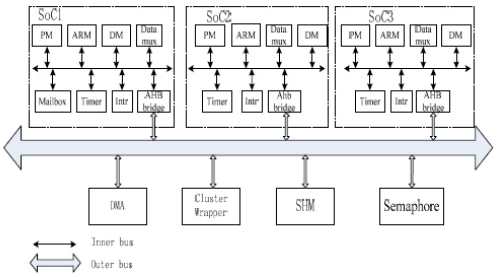

Figure 4. The architecture of the cluster based MPSoC proposed by [12]

The proposed solution is divided into inner buses, which are present in each SoC itself - forming each cluster - and the outer bus, which connects them to each other and to external peripherals.

Still, the work proposed by [4] also targets pure NoC implementation by adding bus-based interface on NoC routers. The main goal is to ease the integration with other bus-based IP components, which are more commonly found. Thus, the proposed NoC has the ability of integrating standard non-packet based components thus reducing design time.

Other approaches also studied the use of buses in NoCs with different purposes [15], [20]. In our case, we still want to use the NoC infrastructure but instead of adding another level of communication we propose to use virtual domains.

Next section introduces some concepts about embedded virtualization.

3 Virtualization and Embedded Systems

First of all, even for classic virtualization concepts, which date back more than 30 years [8], the main component involving virtualization is the hypervisor. It is the hypervisor the responsible for managing the virtual machines (also known as virtual domains) by providing them the needed scenario for its fine work.

To implement the hypervisor, also known as Virtual Machine Monitor (VMM), commonly two approaches are used. In *hypervisor type 1*, also known as *hardware level virtualization*, the hypervisor itself can be considered as an operating system, since it is the only piece of software that works in kernel mode, like depicted in Figure 5. Its main task is to manage multiple copies of the real hardware - the virtual boards (virtual machines or domains) - just like an OS manages multitasking.

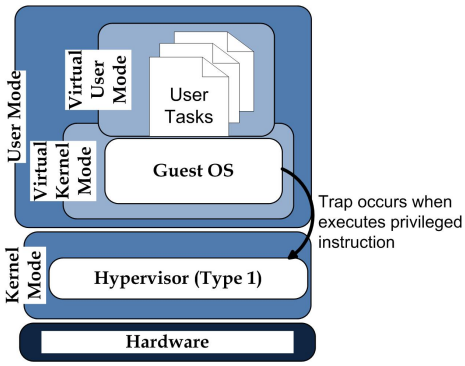

Figure 5. Hypervisor Type 1

Type 2 hypervisors, also known as *operating system level virtualization*, depicted in Figure 6, are implemented such that the hypervisor itself can be compared to another user application that simply "interprets" the guest machine ISA.

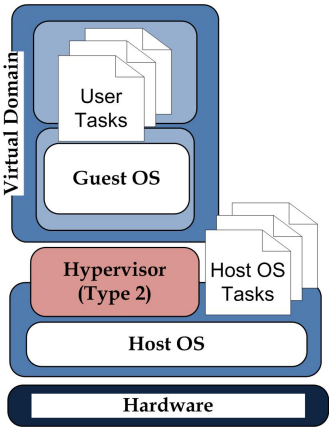

Figure 6. Hypervisor Type 2

One of the most successful techniques to implement virtualized systems is known as *para-virtualization*. It is a technique that replaces sensitive instructions of the original kernel code by explicit hypervisor calls (also known as *hypercalls*). Sensitive instructions belong to a classification for the instructions of an ISA (Instruction Set Architecture) into three different groups, proposed by Popek and Gold-

berg [19]:

1. *privileged instructions*: those that trap when used in user mode and do not trap if used in kernel mode;

2. *control sensitive instructions*: those that attempt to change the configuration of resources in the system, and;

3. *behavior sensitive instructions*: those whose behavior or result depends on the configuration of resources (the content of the relocation register or the processor's mode).

The goal of para-virtualization is to reduce the problems encountered when dealing with different privilege levels. Usually, a scheme referred to as *protection rings* is used and it guarantees that the lower level rings (Ring 0, for instance) holds the highest privileges. So, most of OSs are executed in Ring 0, thus being able to interact directly with the physical hardware.

When the hypervisor is adopted, it becomes the only piece of software to be executed in Ring 0, bringing severe consequences for the guest OSs: they are no longer executed in Ring 0, instead, run in Ring 1, with fewer privileges.

These concepts are present in the virtualization done for general purpose systems but are very important when dealing with embedded systems' typical challenges. Next, some peculiarities found in the application of virtualization solutions in embedded systems are discussed.

3.1 Virtual-Hellfire Hypervisor

There are several hypervisors with embedded systems' focus [22], [10], [21]. In this work, we adopt the Virtual-Hellfire Hypervisor (VHH) [3], part of the Hellfire Framework. The main advantages of VHH are:

- temporal and spatial isolation among domains (each domain contains its own OS);
- resource virtualization: clock, timers, interrupts, memory;
- efficient context switch for domains;
- real-time scheduling policy for domain scheduling;
- deterministic hypervisor system calls (hypercalls).

VHH considers a **domain** as an execution environment where a guest OS can be executed and it offers the virtualized services of the real hardware to it. In embedded systems where no hardware support is offered, para-virtualization tends to present the best performance results. Therefore, in VHH, domains need to be modified before being executed on top of it. As a result, they do not manage hardware interrupts directly. Instead, the guest OS must be modified to allow the use of virtualized operations provided by the VHH (hypercalls).

Figure 7 depicts the Virtual-Hellfire Hypervisor structure. In this figure, the hardware continues to provide the basic services as timer and interrupt but they are managed by the hypervisor, which provides hypercalls for the different domains, allowing them to perform privileged instructions.

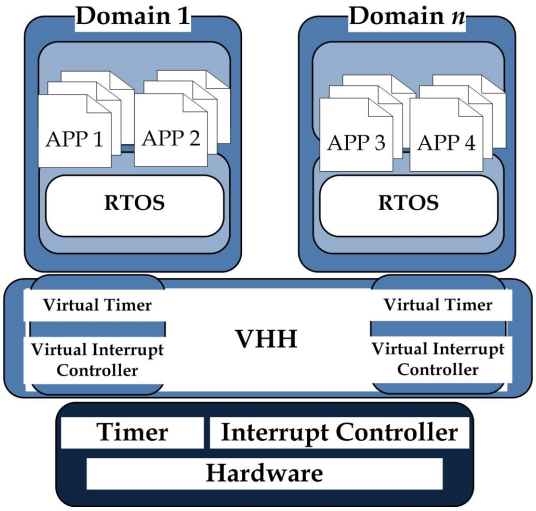

Figure 7. Virtual-Hellfire Hypervisor Domain structure

Thus, Virtual-Hellfire Hypervisor is implemented based on the HellfireOS [1] and counts on the following layers:

- **Hardware Abstraction Layer - HAL**, responsible for implementing the set of drivers that manage the mandatory hardware, like processor, interrupts, clock, timers etc;
- **Kernel API and Standard C Functions**, which are not available to the partitions;
- **Virtualization layer**, which provides the services required to support virtualization and para-virtualization services. The hypercalls are implemented in this layer.

Figure 8 depicts the architecture of the VHH, where some of the following modules can be found:

- *domain manager*, responsible for domain creation, deletion, suspension etc;
- *domain scheduler*, responsible for scheduling domains in a single processor;

- *interrupt manager*, which handles hardware interrupts and traps. It is also in charge of triggering virtual interrupt and traps to domains, and;
- *hypercall manager*, responsible for handling calls made from domains, being analogous to the use of system calls in conventional operating systems.

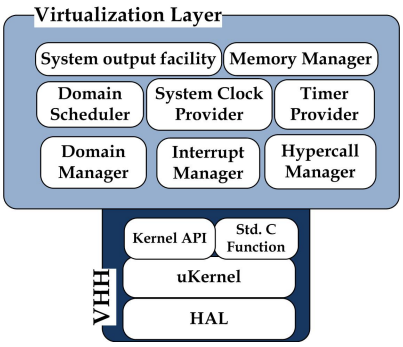

Figure 8. VHH System Architecture

4 Virtual Cluster-Based MPSoCs

This section describes the Virtual Cluster-Based MPSoC proposal. Initially, let us take a look into each cluster of the MPSoC.

Since our work is based on the Hellfire Project, we also use the Plasma [5] processor, a MIPS-like architecture. Therefore, the VHH is placed on a Plasma processor as the basis of our cluster. Then, the VHH is responsible for managing several virtual domains. In our case, each VHH is responsible for managing its own processing cluster and it allows the internal communication of these processors through shared memory.

Figure 9 is divided in two parts. In *A*, the current version for memory division, which only predicts a single memory partition per virtual domain, is shown. This means that this partition is considered to be the local memory for a given virtual domain. In *B*, it is possible to see that an extra partition was added: the *shared* partition. Here, the idea is to provide easy and low overhead communication inside the cluster.

The VHH was extended to allow the communication in two levels. The first level, is named as *intracluster* communication and occurs through shared memory. Currently, this is not user transparent and a specific hypercall must be used for this communication. In this hypercall, a single CPU identification (CPU_ID) must be used, which means they belong to the same processing cluster.

These hypercalls are similar to the communication functions provided by the HellfireOS and have the follow-

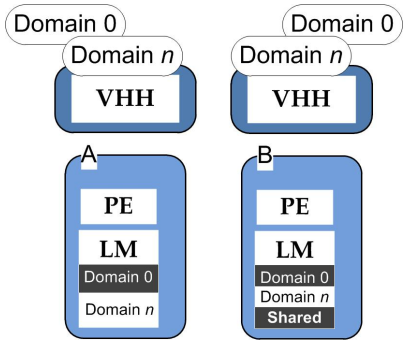

Figure 9. VHH Memory for (A) Non-clustered systems (B) Clustered systems

ing parameters: VHH_SendMessage (*cpu_id, task_id, message, message_length*) used to send a message through the shared memory and VHH_ReceiveMessage (*source_cpu_id, source_task_id, message, message_length*) used to receive it.

The second communication level is done among clusters, through the NoC. In our case, we use the HERMES NoC [17] and a MIPS-like processor in each router. We adopted a Network Interface (NI) as a wrapper which connects the NoC router to the processor located in its local port. This interface, works in a similar way that the non-virtualized approach. This increases the possibility of using several NoC infrastructures as the underlying architecture. Figure 10 depicts this approach.

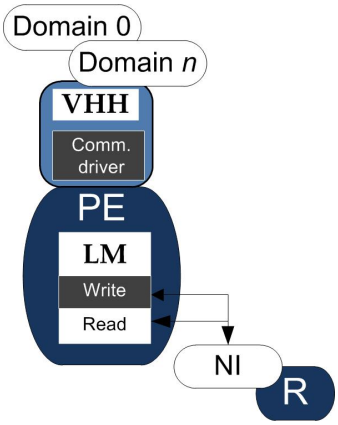

Figure 10. VHH Communication Infrastructure with NoC based Systems

The wrapper is connected through specific memory addresses: read and write, to the Plasma. Still, a communication VHH driver had to be written to allow the integration between the wrapper and the virtual cluster. Also, the hypercalls provided by the VHH allow a virtual processor to send or receive messages with an extra parameter: the Vir-

tual CPU ID, as an identification of the virtual CPU on a specific cluster.

Thus, the hypercalls to be used to the inter-cluster communication are: VHH_SendMessageNoC (*cpu_id, virtual_cpu_id, task_id, message, message_length*) used to send a message through the NoC and VHH_ReceiveMessage (*source_cpu_id, source_virtual_cpu_id, source_task_id, message, message_length*) used to receive it.

The complete vision of the system is depicted in Figure 11. In the Figure, VHH is the Virtual Hellfire Hypervisor. LM stands for Local Memory. NI stands for Network interface and PE, for Processing Element. R represents each router of the NoC.

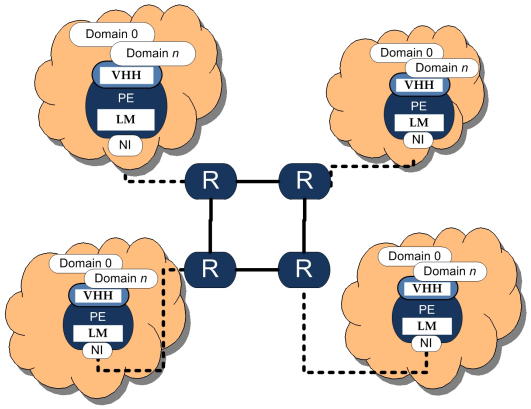

Figure 11. Virtual Cluster-Based MPSoC proposal

5 Use Cases and Experimental Results

In this section we highlight some possible use cases for Cluster-Based MPSoCs and some preliminary prototyping results.

The main use for a Cluster-based MPSoC is the possibility for field specialization. In this case, each cluster is responsible for executing a set of tasks with a common purpose. For instance, it is possible to execute a JPEG decoder in one cluster, a MPEG decoder in another and so on. In this case, the greatest advantage is to simplify the communication of similar tasks, since they share a given memory area, but still allowing a great number of processors, increasing system scalability through the NoC usage. Figure 12 depicts an example of cluster-based MPSoC with application specialization.

Another possible use of the Virtual Cluster-based MPSoC is when decreasing area with guaranteed system scalability is needed. Scalability is assured by NoC usage and the cluster-based MPSoC itself allows an easier use of real-

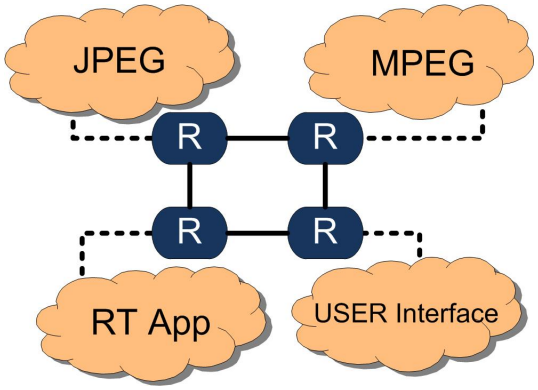

Figure 12. Virtual Cluster-Based MPSoC with Application Specialization

time tasks with no extra communication penalties. Regarding area occupation, we prototyped some possible configurations to illustrate the benefits of our approach in this issue. We used the Xilinx Virtex-5 XC5VLX330T FPGA.

First, when using the HellfireOS with a Plasma processor, we usually indicate a processor with at least 16KB of local memory. HellfireOS is a much optimized kernel and depending on the application even such a small memory can fulfill the expected needs. When using the VHH, more memory is required and the total memory size depends especially on the number of virtual domains that are required. Although greater memory sizes infer more block RAMs, it does not affect the FPGA area measured in LUTs. In all experiments performed, the total system memory could be inferred as block RAMs.

We used three different MPSoC configurations, all with 16 processors (physical or virtual). First, we have a 16 processor MPSoC, distributed in a 4x4 NoC where each router carries its own processor, known as *Pure 4x4 NoC* approach.

The second MPSoC configuration regards a 2x2 NoC with bus-based clustering system, known as *Bus Clustered* approach. Here, each router has a wrapper to connect it to the clustered-bus, and each bus carries four processors.

Finally, the last approach is the Virtual cluster-based (*V-Cluster 2x2 NoC*) where a 2x2 NoC was used again and each router contains a single physical processor. This processor runs the VHH, where 4 virtual domains are emulated per cluster, totalizing the 16 processors of the MPSoC.

In the first two solutions, each processor has 16KB of local memory. The last, for the virtual cluster approach, 4 processors with 128KB of memory each were employed. In Table 1, it is possible to see the prototyping results for three different MPSoCs.

These results show a decrease of the area occupation in up to 70%, depending on the processor local memory con-

Table 1. Area results for MPSoCs configuration

Configuration	Area occupation (LUTs)
Pure 4x4 NoC	60934
Bus Clustered 2x2 NoC	56099
V-Cluster 2x2 NoC	17179

figuration and the original MPSoC configuration. Also, depending on the bus structure used for the Bus-based clustered version, the bus communication overhead is similar to the virtualization overhead.

6 Concluding Remarks and Future Work

This paper presents a new proposal for MPSoC configuration using virtualization with a cluster-based approach. For validation purposes, we use an extension of the Hellfire Framework and, in order to incorporate our virtualization methodology, the Virtual-Hellfire Hypervisor (VHH).

We use a HERMES NoC as the underlying architecture where each processor runs the VHH, forming the processing clusters. We achieved up to 70% decrease in FPGA area occupation in our preliminary tests.

As a future work we intend to get comparison results for performance and overheads with other approaches. Still, we want to improve the proposal itself, especially regarding memory and I/O management.

Acknowledgment

The authors acknowledge the support granted by CNPq and FAPESP to the INCT-SEC (National Institute of Science and Technology Embedded Critical Systems Brazil), processes 573963/2008-8 and 08/57870-9. Also, this work is supported in the scope of the project SRAM by the Research and Projects Financing (FINEP) under Grant 0108031000.

References

[1] A. Aguiar, S. Filho, F. Magalhaes, T. Casagrande, and F. Hessel. Hellfire: A design framework for critical embedded systems' applications. In *Quality Electronic Design (ISQED), 2010 11th International Symposium on*, pages 730 –737, 2010.

[2] A. Aguiar and F. Hessel. Embedded systems' virtualization: The next challenge? In *Rapid System Prototyping (RSP), 2010 21st IEEE International Symposium on*, pages 1 –7, 2010.

[3] A. Aguiar and F. Hessel. Virtual hellfire hypervisor: Extending hellfire framework for embedded virtualization support. In *Quality Electronic Design (ISQED), 2011 12th International Symposium on*, 2011.

[4] B. Ahmad, A. Ahmadinia, and T. Arslan. *Dynamically Reconfigurable NoC with Bus Based Interface for Ease of Integration and Reduced Design Time*. IEEE, June 2008.

[5] O. Cores. Plasma most mips i(tm) opcodes. http://www.opencores.org.uk/projects.cgi/web/mips/, Accessed, September 2009, 2007.

[6] I. Corp. Intel ixp2855 network processor. Web, Available at http://www.intel.com/., 2005.

[7] L.-F. Geng. Prototype design of cluster-based homogeneous Multiprocessor System-on-Chip. *2009 3rd International Conference on Anti-counterfeiting, Security, and Identification in Communication*, pages 311–315, Aug. 2009.

[8] R. P. Goldberg. Survey of virtual machine research. *Computer*, pages 34–35, 1974.

[9] J. Goodacre and A. N. Sloss. Parallelism and the arm instruction set architecture. *Computer*, 38:42–50, July 2005.

[10] G. Heiser. Hypervisors for consumer electronics. pages 1 –5, jan. 2009.

[11] A. Jerraya, H. Tenhunen, and W. Wolf. Multiprocessor systems-on-chips. *Computer*, 38(Issue 7):36– 40, July 2005.

[12] X. Jin, Y. Song, and D. Zhang. *FPGA prototype design of the computation nodes in a cluster based MPSoC*. IEEE, July 2010.

[13] M. Kistler, M. Perrone, and F. Petrini. Cell multiprocessor communication network: Built for speed. *IEEE Micro*, 26:10–23, May 2006.

[14] A. Ltd. Nios ii processor reference. Web, Available at http://www.altera.com/., 2009.

[15] R. Manevich, I. Walter, I. Cidon, and A. Kolodny. *Best of both worlds: A bus enhanced NoC (BENoC)*. IEEE, Nov. 2010.

[16] G. D. Micheli and L. Benini. *Networks on Chips: Technology and Tools (Systems on Silicon)*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2006.

[17] F. Moraes, N. Calazans, A. Mello, L. Möller, and L. Ost. Hermes: an infrastructure for low area overhead packet-switching networks on chip. *Integr. VLSI J.*, 38(1):69–93, 2004.

[18] T. Noergaard. *Embedded Systems Architecture: A Comprehensive Guide for Engineers and Programmers*. Newnes, 2005.

[19] G. J. Popek and R. P. Goldberg. Formal requirements for virtualizable third generation architectures. *Commun. ACM*, 17(7):412–421, 1974.

[20] T. Richardson, C. Nicopoulos, D. Park, V. Narayanan, Y. Xie, C. Das, and V. Degalahal. A hybrid soc interconnect with dynamic tdma-based transaction-less buses and on-chip networks. In *VLSI Design, 2006. Held jointly with 5th International Conference on Embedded Systems and Design., 19th International Conference on*, page 8 pp., 2006.

[21] W. River. Wind river. Web, Available at http://www.windriver.com/. Accessed at 2 oct., 2010.

[22] XEN.org. Embedded xen project. Web, Available at http://www.xen.org/community/projects.html. Accessed at 10 ago., 2010.