Distributed Approaches for Self-Adaptive Embedded Systems

ERSA'12 Academic Invited Paper

Pascal BENOIT
LIRMM, UMR 5506, CNRS University of Montpellier, 161 rue Ada, 34095 Montpellier Cedex 5, France

Abstract - *In the recent years, there has been a growing interest in self-adaptive embedded systems. Compared to the conventional approach, they require a control loop based on a three-step process: (1) observation, handled by a set of sensors/monitors, (2) diagnosis, which analyzes observed data to adapt and optimize the system, and (3) action, which tunes system parameters accordingly. Putting an additional intelligence into the circuit so that it is capable of modifying itself a set of parameters is not a new idea. But today, it seems that the conditions have been met to build such circuits. Firstly, self-observation has been made feasible with different kind of monitors, like activity counters, temperature sensors, critical path-monitors, etc. Secondly, it is possible to tune the voltage/frequency pairs, to migrate the code of a given task from one processing element to another, to adapt the routing of data in the interconnection network, etc. So what is the real challenge today? Achieving a complex but realistic unified self-adaptation mechanism, which strikes the balance between the introduced overhead, power consumption, performance and area. Given the increasing complexity of embedded systems, our approach is to consider a regular distributed architecture, with a set of identical Processing Elements, interconnected with a network on chip. Thus, all the hardware/software building blocks required for self-adaptation, are the same for each PE, which simplifies the scalability for future technologies. During this talk, we will present an open experimental platform and original approaches for the control loop based on the three-step adaptation process; we will analyze the cost of their implementation and will draw the perspectives offered by such techniques.*

Keywords: MPSOC, Distributed Architectures, Self-Adaptability, Game Theory, Consensus Theory

1 Introduction

For many years, the evolution of silicon technologies has pushed the emergence of new high-tech products, leading to new applications and new uses. Today, the application needs seem to clearly lead the technological developments. The ever-increasing performance and low-power requirements continuously brings new challenges in the semiconductor research and industry communities. In the 2011 edition of the ITRS [1], it was reported that in the next 10 years, the processing power will increase by 1000x for SOC Consumer Portable devices. As depicted in figure 1, the number of processing engines, logic and memory size will exponentially grow. Due to the short time-to-market and reduced lifecycles, the design efforts cannot be increased: it is assumed that in 2020, 90% of a SOC will be a reused design (58% in 2012).

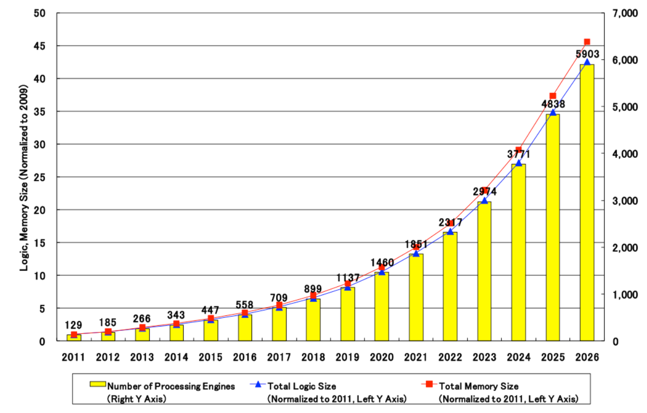


Fig. 1. SOC Consumer Portable Design Complexity Trends from ITRS

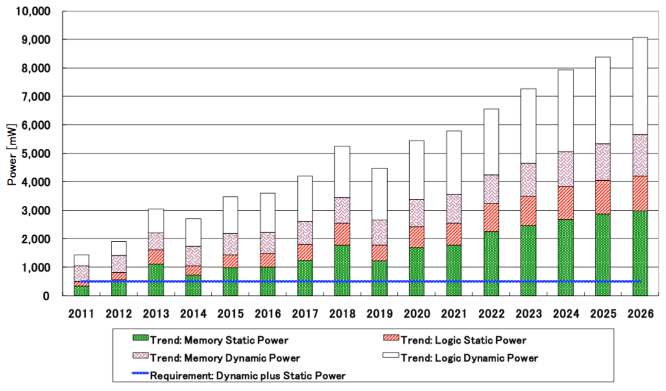


Fig. 2. SOC Consumer Portable Power Consumption Trends from ITRS

In this context, energy is also a critical issue. The power budget for a portable device ranges from 0.5 to 2W. It is clear from figure 2, that the current trend will not be acceptable for future circuits. The reduction of the power consumption of electronic products must be addressed at all the stages: technology, design and applications.

From a technological point of view, variability has become a major issue. While process variations impact process, supply voltage and internal temperature [2], chip performances are also dependent on environmental and on applicative changes that may further influence chips behavior [3].

Performance, energy efficiency, technological variability and application versatility motivate the need of self-adaptability. The objective is to develop a system able to adapt itself to new applicative requirements as well as changes in the chip itself, or in its environment. As an example, the available energy for a telecom application is strongly reduced under weak battery conditions on mobile terminals, while real-time constraints depend on the telecom configuration. Compared to the classical approach, a self-adaptive system requires a control loop based on a three-step process: (1) observation, handled by a set of sensors/monitors, (2) diagnosis, which analyzes observed data to adapt and optimize the system, and (3) action, which tunes system parameters accordingly. The real challenge is to achieve a complex but realistic unified self-adaptation mechanism, which strikes the balance between the introduced overhead, power consumption, performance and area. Given the increasing complexity of embedded systems, our approach is to consider a regular distributed architecture, with a set of identical Processing Elements, interconnected with a network on chip. Since 2005, we develop our own MPSOC platform and investigate different distributed strategies for the monitoring and the optimization at run-time.

The paper is structured as follows. In the next section, we will give a definition of a distributed self-adaptive system. Then, we will present the platform developed at LIRMM called Open-Scale. In section 4, we will summarize our research works on monitoring techniques. Optimization techniques handled by distributed controllers will be reported and analyzed in section 5. Finally, we will conclude this article and give some future research directions.

2 A definition of a distributed self-adaptive system

The objective of this work is to find coherent solutions for the design of integrated systems that are efficient, low power, insensitive to technological process variations, reliable, scalable, with capabilities enabling them to adapt to their environment, and to the various applications they must support. We believe that simplicity and regularity of the system architecture are two key aspects that should guide the design choices. We present in this section on the one hand the concept of self-adaptability applied to embedded systems, and on the other hand, the specifications of the target architecture and the means to implement this system to make it adaptive.

2.1 Self-adaptability concept

Self-adaptability is the faculty attributed to an entity that can be self-sufficient and act within its environment to optimise its functions. Self-adaptability also describes a system that can manage itself using its own rules.

In order to apply this concept to the reality of technological systems, this study will start with the abstract view of a system architecture while applying the notion of self-adaptability. Figure 3 gives a synthetic view of self-adaptability: the activator creates the physical state of the system and the diagnosis motivates it. Self-adaptability can be used to lower energy consumption in microelectronics. In robotics, the challenge is to increase performance in different environments. In the artificial intelligence domain, self-adaptability is the consequence of a life cycle where the sensors observe the system, the diagnosis gives the direction, the language of command orders (decisions) and actuators act.

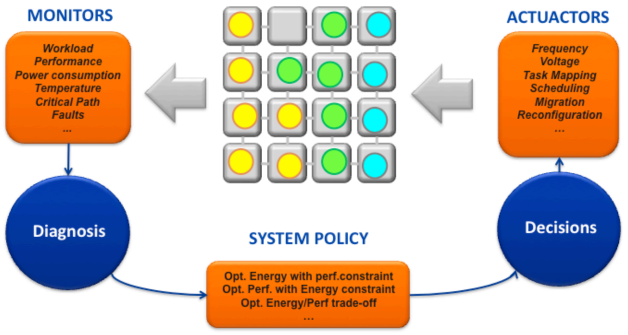


Fig. 3. Self-adaptability

2.2 System Specifications

When referring to current integrated systems, it is often about MPSOC or multi-core system. This is in fact for both integrated systems consisting of several processing units. When these ones are identical programmable general-purpose processors, MPSOC are said to be homogeneous. When dealing with heterogeneous architectures, there are different kinds of computing resources: general-purpose processors to support the system management, but also DSP, dedicated accelerators, etc. The current trend is finally the same for over 40 years: the integration density doubles approximately every 2 years, we then expect in the longer term Many Core systems, or MP2SOC (Massively Parallel MPSOC), *i.e.* systems with hundreds or thousands of processing units. Obviously, this complexity poses a number of questions about the evolution of design methods, verification, manufacturing, test, programming, debugging, etc.

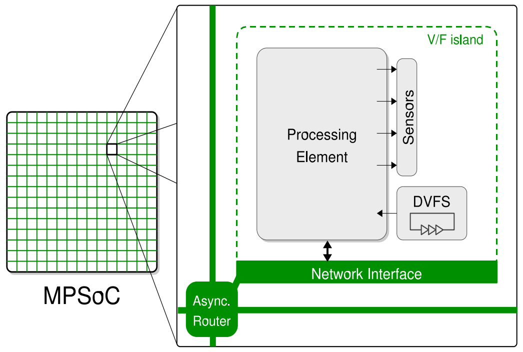


Fig. 4. System overview.

Given the huge design space due to the increasing number of transistors, it is not possible to build a 10 billion-transistor circuit from scratch, for each new product. During the 2000s, IP reuse and platform based design appeared as new methodologies to accelerate the Time To Market for SOCs. Due to the ever-increasing complexity, we start hearing about (Sub-) System IPs Reuse to build the future many core systems.

By observing these changes, we decided in 2005 to devote our research efforts on designing a regular and homogeneous MPSOC architecture. The basic idea is to have a simple subsystem, composed of a programmable processing element, memory, and a generic interconnection module, that we can instantiate theoretically to infinity. The primary version of our architecture called HS-Scale was first published in [4]. The thrust of this project is to define a generic and regular software and hardware support, to facilitate scaling. This very flexible model is originally not designed for specific applications, but with the addition of programmable and/or dedicated accelerators to the building block, one can think advantageously taking advantage of the same principle of regularity for specific products.

The regular architecture of our MPSOC is described in Figure 4. The PEs are interconnected by a Network-on-Chip (NOC) [5-8]. A NOC is composed of Network Interfaces (NI), routing nodes and links. NI implements the interface between the interconnection environment and the PE domain. It decouples computation from communication functions. Routing Nodes are in charge of routing the data between the source and destination PEs through links. Several network topologies have been studied in the literature [9, 10]. Our approach is based on a 2D mesh interconnect. We consider that the offered communication throughput is enough for a general purpose application set. Our NOC fulfills the "Globally Asynchronous Locally Synchronous" (GALS) concept, by implementing asynchronous nodes and asynchronous-synchronous interfaces in NIs [11, 12]. As in [13], GALS properties allow MPSOC partitioning into several Voltage Frequency Islands (VFI). Each VFI contains a PE clocked at a given frequency and voltage. This approach allows real fine-grain power management.

Dividing the circuit into different power domains using GALS must facilitate the emergence of more efficient designs that take advantage of fine-grain power management [14]. As in [15, 16], the considered MPSOC incorporates distributed Dynamic Voltage and Frequency Scaling (DVFS): each PE represents a VFI and includes a DVFS device. It consists of adapting the voltage and frequency of each PE in order to manage power consumption and performance. A set of sensors integrated within each PE must provide information about the power consumed, the local temperature, the performance or any other metric needed to manage the DVFS.

3 Open-Scale Prototyping Platform

Since 2005, we develop an MPSOC architecture based on the principles described in the previous section, *i.e.* a regular and distributed architecture for the implementation and the evaluation of self-adaptability principles. This architecture published for the first time in [4] has undergone a number of developments and changes. The current version of this architecture is called Open-Scale: it is therefore an MPSoC governed by the basic principles outlined above. We have different models of the architecture: a version based on ISS (Instruction Set Simulator) and SystemC, and a second based on the fully synthesizable RTL code. The first one allowed making high level explorations, and the development of the RTOS. Although the architecture is the same for a system standpoint, we are interested especially here in the RTL version, validated on multiple FPGA prototypes, as the one depicted in figure 5. In this section we describe the hardware and software building elements of the Open-Scale platform.

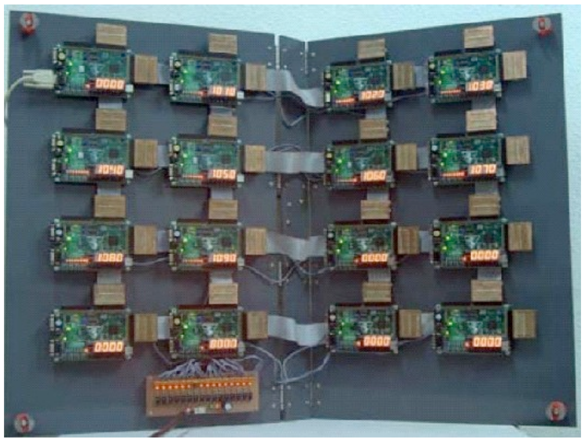


Fig. 5. MPSOC prototype

3.1 NPU, the Network Processing Unit

The Open-Scale system is an architecture that employs a distributed memory/message passing approach and a 2D-mesh NOC for connecting different PEs. As the PEs are connected through a network, they are called Network Processing Units (NPUs) [4].

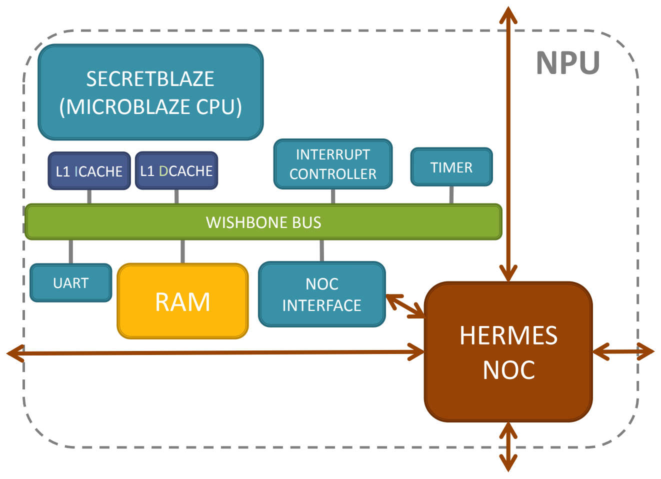


Fig. 6. NPU architecture overview

The figure 6 depicts the internal architecture of the NPU, which includes: (*i*) a general purpose processor, the SecretBlaze [17,18], (*ii*) an embedded RAM, (*iii*) an interrupt controller, (*iv*) a timer, a UART, (*v*) a NI, a (*vi*) HERMES-based router [19] and a (*vii*) Wishbone v4 bus [20].

Originally designed with the Plasma processor [21], the NPU is now based on the SecretBlaze [17,18], which is a highly configurable open-source RISC soft-core processor developed by our research group. It implements the MicroBlaze instruction set architecture with a MIPS five-stage pipeline, where most of the instructions require one clock cycle to be executed. As reported in [17], it achieves the same performance level as the MicroBlaze for FPGA implementations. This processor was developed with a modular approach, not only to ensure reliability and efficiency across the whole design, but also to provide better design reuse opportunities in various research and educational projects. It is available as a single open-source processor at *http://www.lirmm.fr/openscale*

The flexibility is one of the driving aspects of the Secret- Blaze design. On the one hand, the core provides several optional logical and integer instructions such as multiplication, division, and pattern operations, which balances computing performance and area cost to meet embedded system requirements. On the other hand, the SecretBlaze is a MMU less processor with a simplified memory sub-system that offers optional configurable data and instruction caches. It implements the pipelined Wishbone protocol for memory interfaces. However, no global cache coherency between NPU is provided.

The SecretBlaze uses an embedded RAM as local memory. The interrupt controller can handle up to 8 interrupts with masking, arming, and poling mechanisms. The timer is a 32-bit counter that can generate an interrupt according to a configurable time window. Besides, a UART interface, which is adjustable via software, can be used for debugging purposes. These components are interconnected by a Wishbone v4, which is a standard and an open-source bus [20]. The communication between the NPU and the NOC router is implemented in the NI (Network Interface), which defines HW/SW integration (*e.g.* bus width, bandwidth), as well as packing/unpacking the packets from/to the NOC.

The adopted NOC router is a small XY router based on HERMES [19]. The NOC employs packet switching of wormhole type: the incoming and outgoing ports used to route a packet are locked during the entire packet's transfer. The routing algorithm is an XY engine that allows deterministic routing. Each router processes one incoming FIFO per port. The size of FIFOs can be chosen regarding the desired communication performance.

3.2 Open-Scale RTOS

Open-Scale is a set of programmable Network Processing Units. Programming (and debugging) complex applications tp make an efficient use of multiple computing units is a real challenge. It is necessary to provide a coherent middleware layer to simplify these two fundamental aspects of embedded systems. In order to keep a distributed memory structure and to preserve the scalability of the system, each NPU operates asynchronously and communicates with each other by means of a Messaging Passing Interface (MPI) protocol. The global operation is performed in a distributed fashion and no global shared-memory is used.

Each NPU runs a tiny preemptive RTOS that was further extended from the Plasma RTOS [21]. The global structure of the operating system is depicted in figure 7. The RTOS provides a multi-threaded preemptive execution, using a scheduler based on thread priorities that is executed periodically according to a fixed *timeslot*. A round robin scheduling algorithm is executed when all tasks have the same priority. The structure of this system is divided into 4 categories: (*i*) services that provide the basic operating system requirements, (*ii*) communication, (*iii*) drivers, and (*iv*) libraries. The RTOS allows the use of semaphores and mutexes, communication between local and remote tasks, and dynamic memory allocation, as well. Further, it also provides the standard C library together with a compact math library that allows floating point operations as well as software multiplications/divisions. Different kind of drivers, e.g. timer, UART, etc., are also available.

Due to the distributed memory characteristic of the system, the applications are described using Kahn Process Network (KPN) formalism [22].

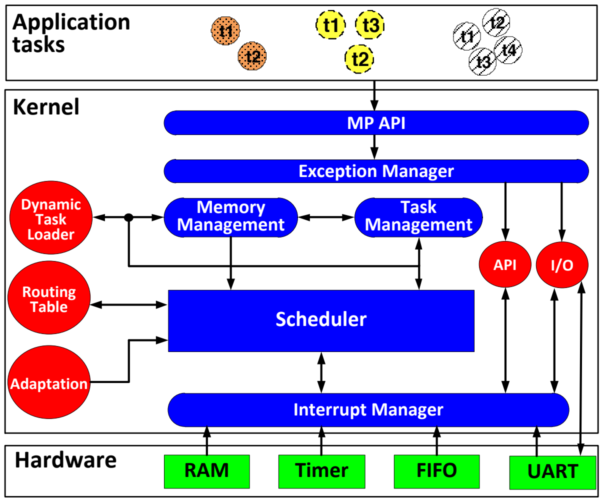


Fig. 7. Structure of the Open-Scale RTOS

MPI provides a comprehensive number of primitives that relate to general-purpose distributed computing; a number of works have devised lightweight implementations supporting only a subset of MPI mechanisms for embedded processors and systems. This makes sense since KPN formalism offers a sufficient support that requires only blocking read operations, which are necessary to model, for instance, data flow (*e.g.* video and audio) applications. Some MPI implementations are layered, and advanced communication synchronization primitives (*e.g.* collective) found in the upper layers make use of the simple point-to-point primitives such as *MPI Send()* and *MPI Receive()*. This enables using these collective mechanisms in an application-specific basis in case they prove necessary. The KPN computation model allows deterministic behavior of the application in an asynchronous way. Furthermore, tasks placement can be optimized depending on the user requirements (*e.g.* computation time, energy consumption).

The Open-Scale RTOS was implemented in such a way that users can easily choose which features are needed in their implementation in order to either save memory or meet performance requirements. In this scenario, new services and features were implemented in order to be compliant with the SecretBlaze architecture, while providing more efficiency in terms of management and QoS support (Table 1).

Table 1 summarizes some services that were included in the Open-Scale RTOS. As mentioned before, one of the goals of Open-Scale is to explore adaptive mechanisms (*e.g.* monitoring techniques, distributed control, dynamic voltage and frequency scaling, task migration, etc.). For instance, to enable dynamic load balancing, the system has to be able to move running tasks from one NPU to another. For that reason, a run-time loading mechanism was included to allow compiled separately applications from the RTOS being dynamically uploaded at run-time.

Table 1. Services Included into the Open-Scale RTOS

Plasma RTOS Services	**Open-Scale Services**
1- generation of a single object file;	1- run-time dynamic applications loading;
2- preemptive round-robin sheduler based on thread priorities;	2- preemptive round-robin scheduler based on thread credits;
3- intra-NPU communication based on local FIFOs;	3- intra-NPU communication based on messages exchanged by software FIFOs;
4- Extra-NPU communication (e.g CP protocol) through ethernet;	4- Extra-NPU communication (RAW protocol was included), as well as *MPI_Send* and *MPI_Receive*;
5- interrupt and exception handling;	5- run-time monitoring support;
6- dynamic memory allocation and deallocation;	6- decision-making mechanisms;
7- queues, semaphores, mutexes.	7- a run-time control system used for regulating NPU frequency;
	8- API with new primitives, etc;
	9 – development of new drives;
	10- dynamic mapping heuristics.

Besides, a preemptive round-robin scheduler based on thread credits has been implemented, avoiding task execution starvation. Moreover, intra/extra-NPU communications were extended to provide more flexibility and system performance. For example, the RAW protocol was implemented in order to achieve better performance when compared to TCP/UDP (as shown in [23]). Further, online system-monitoring mechanisms were included, in order to access hardware monitors available, or software defined monitors (CPU load, (*ii*) FIFO load, etc.). Once monitored information is provided, online decisions can be taken by decision-making mechanisms, like a run-time control system used for regulating NPU frequency. All the middleware support for self-adaptation (*e.g.* local DVFS control, optimization, etc.) was included to provide a complete infrastructure for self-adaptation strategies investigations.

4 Distributed Monitors

We have defined a set of software and hardware components, based on distributed resources and a principle of regularity of the architecture that has led us to the design of the Open-Scale platform. In this section, we will focus on the monitoring aspect. To achieve a self-adaptive system, it is necessary to provide a set of sensors allowing the system to observe both its internal behavior and its environment. We focus here on internal monitoring.

We seek solutions to evaluate strategies of self-adaptation that are also compatible with an FPGA prototyping environment. We therefore propose in this section to explore several types of approaches to monitoring in this context, by focusing on hardware sensors, such as PVT sensors or activity counters, on software monitors directly integrated into the middleware of Open -Scale, and finally an integrated database enabling a coherent organization for a subsequent operation for system optimization.

4.1 PVT sensors

Monitoring with trustable and valuable information systems made of a billion transistors at a reasonable area and performance cost is a real challenge, because of the increasing random nature of some process parameters, the spatial dependence of process (including aging), voltage and temperature variations, but also the broad range of time constants characterizing variations in these physical quantities.

There is a vast literature on this topic, and different kind of approaches. Several PVT sensors commonly used for post fabrication binning have been investigated [24-29] for global variability compensation. They are based on specific structures like Ring Oscillators or Replica Paths to measure, at runtime, the physical and electrical parameters required to dynamically adapting the system, *e.g.* operating frequency, the supply voltage, threshold voltage, substrate biasing, etc. Another approach is to directly monitor sampling elements of the chip (Latches or D-type Flip Flop) to detect delay faults. This can be achieved by inserting specific structures or using ad-hoc sampling elements [28-29] to detect a timing violation by performing a delayed comparison or by detecting a signal transition within a given time window.

In the Open-Scale platform, one objective is to design a set of sensors in order to monitor locally the Temperature, Voltage, and Process. The basic idea is to use digital hardware sensors designed with internal resources of the FPGA, *i.e.* configurable logic blocks and switch matrices. In this context, we proposed, designed and compared two structures: a Ring Oscillator and a Path Delay sensor.

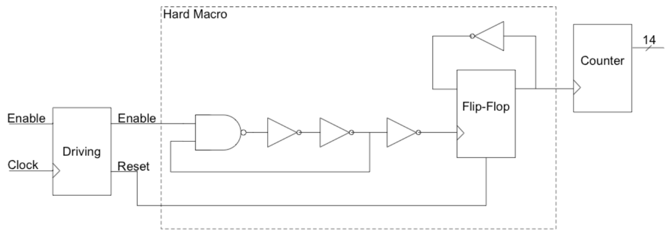


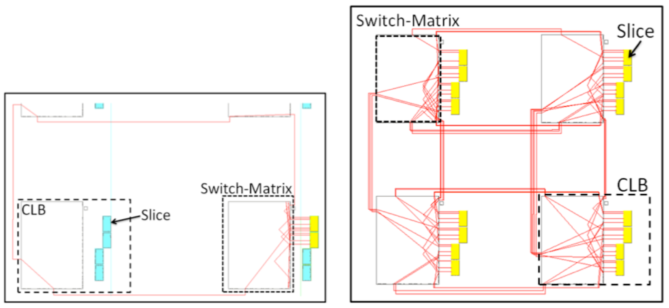
Fig. 8. Ring Oscillator sensor

The Ring Oscillator sensor is based on the measurement of the oscillator frequency. For instance in [30], this structure was used as an internal temperature sensor for FPGA. The frequency of a ring oscillator is measured and converted into temperature. We have developed a new version of this kind of sensor in Open-Scale. Its structure is depicted in Figure 8. The main part of the sensor is a *2p+1* inverter chain. The oscillation frequency directly depends on the FPGA process performance capabilities, for a given voltage and temperature. The first logic stage enables the oscillator to run for a fix number of periods of the main clock. The ending flip-flop is used as a frequency divider and allows filtering glitches from the oscillator. The final logic stage counts the number of transitions in the oscillator and transmits the count result. Then, the count result is used to calculate the oscillator frequency as follows:

$$F = \frac{count * f}{p}$$

where F is the ring oscillator frequency, *count* is the 14-bit value of the counter, *f* is the operating frequency of the clock and *p* is the number of enabled clock periods for which the sensor is active.

In order to use this sensor for resource monitoring in FPGA, a three-inverter ring oscillator was implemented. With this configuration, the core of the sensor (ring oscillator + first flip-flop) only consumes 4 LUTs. A Hardware Macro was designed so that the very same sensor structure can be mapped at each FPGA location (Fig. 9(a)). It possibly allows characterizing separately each CLB of an FPGA.



(a) Ring Oscillator (b) Path Delay

Fig. 9. Hard-Macro implementation of both PVT sensors

It exists a lot of techniques [31] to manage Critical Path but very few are used in FPGAs as PVT sensors. The Path Delay Sensor proposed here is directly inspired by CPM. Its structure is depicted in Figure 10. The idea of the Path Delay Sensor is to adapt CPM to FPGA. Indeed, the regularity of the FPGA structure enables to create more easily a critical path replica in FPGA than in ASIC.

The Path Delay Sensor is composed of *n* LUTs and n flip- flops (FF). The LUTs are chained together and a FF is set at the output of each LUT. A clock signal is applied to the chain and is propagated into the LUT. At each rising edge, a *n-bits* thermometer code is available at the output of FFs. This thermometer code is representative of LUTs and interconnects performances.

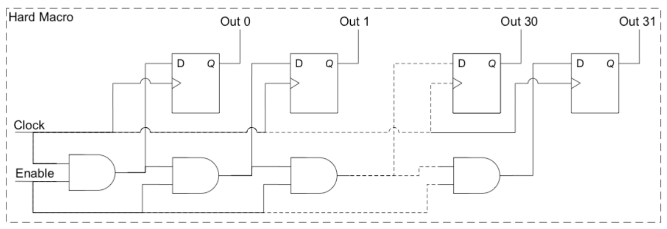


Fig. 10. Path Delay sensor

When the sensor is running, a thermometer code is stored in the FF, and then analyzed. Assuming the code is ”11111111111111000000000000001111”, then the position Nz of the last 0 is identified. The time T required to go through one LUT and the associated interconnect is approximated as follows:

$$T = \frac{Nz + 2}{f}$$

where f is the frequency of the clock signal applied to the sensor. In order to obtain relevant information, the size of this sensor must take into account the FPGA family in which it operates. For example, for a Spartan-3 device, this sensor is composed of 32 stages. It allows propagating a complete period of the Spartan-3 reference frequency clock (50MHz). The figure 9(b) shows the Hard Macro integration of this sensor.

Our study in [32] has proved that both structures were efficient for fast PVT local monitoring in FPGA devices. The ring oscillator is an interesting structure for fine grain monitoring, whereas the Path Delay Sensor will be a preferred structure to allow rapid performances estimations with a minimal area overhead.

4.2 Activity Counters

In some cases, PVT sensors might be not enough accurate and fast to reflect the exact load of the core, especially to estimate the power consumed locally by the Processing Elements. In-situ events counters have been proved to be efficient solutions for monitoring at real-time the activity of a core. They were first developed to better understand application behavior, facilitate debugging and so to improve high-end processors execution flow at run-time. For this usage, they are also called performance counters. These probes are a set of in-situ registers used to record the activity (*i.e.* number of events on some selected signals) in the processor. Significant events having an architectural meaning are typically used, such as instruction execution, floating point operation, memory transactions, pipeline stalls, etc. For instance, the super processor BlueGene/P from IBM is providing 52 counters [33], which can be configured and read through a dedicated API. The ARM processor architecture contains two event counters with an extending instruction set to configure their inputs. Moreover, a dedicated monitoring co-processor can be added through this interface. As the required precision becomes more and more important, the number of events tends to increase. Temporal multiplexing has been proposed to reduce the hardware overhead when applications show periodic execution time [34]. However, gathering information is problematic and may penalize applications with hard performance constraints. As a result, the number of monitors as well as the size of their associated counters must be reduced.

The use of activity counters was extended to embedded system run-time management in [35,36]. In such systems, area constraints hardly limit the possible number of monitors compared to general-purpose processors. Activity counters were generally selected manually and their usage was quite limited. Moreover, assuming that the NPU of Open-Scale is composed of different kind of resources (the processor, memory, peripherals, network interface) that could also be further augmented with accelerators or specialized DSP, it is necessary to provide other activity counters to estimate at run-time the power consumed by the different blocks. In such a context, the selection process of the signals to monitor might become a difficult and time-consuming work.

We tackled this issue by proposing an automated selection of events to be monitored starting from a power model, and usable for any IP. Moreover, our selection process allows striking the balance between the area taken by the monitors and the precision of the model.

Power consumption P_T can be approximated, over the period T, by a linear function of the following form:

$$P_T = c + \alpha_1 N_{e_1} + \ldots + \alpha_i N_{e_i} + \ldots + \alpha_n N_{e_n}$$

where :

- $\{e_i\}_{i=1:n}$ denotes events to which counters will be connected;
- N_{ei} is the number of occurrences of event e_i during the sampling period T;
- α_i is the regression coefficient describing power contribution of event e_i;
- c is a constant term representing static consumption;
- n measures the power model complexity.

Our goal is to minimize the cardinal of the set $\{e_i\}_{i=1:n}$ of events, so that we can limit the model complexity. We propose to use a step-by-step selection technique to detect influential events on power.

Fig. 11 depicts an overview of the proposed modeling flow based on four steps: instrumentation, data processing, model extraction and physical placement of monitors. Our flow starts with a post-synthesis simulation to generate a VCD (Value Change Dump) file containing the transitions at each net over time. At this stage, we try to track highly power consuming blocks, so for the synthesis process we preserve the hierarchical structure of the design. The VCD is used as input to the *Xpower* tool from Xilinx to extract instantaneous power values. In parallel to this, the RTL model of the design is simulated. Events occurring on each controlling signal are time-stamped and then reported, this can be easily done with some options in *Modelsim.*

Our method based on stepwise process was evaluated to select events able to estimate the power consumption. This approach helps to explore cost and accuracy trade-offs when designing such probes. Sampling impact can be easily analyzed along with the number of counters that should be deployed. A control unit extracted from a dedicated VLIW accelerator for Open-Scale was used as a case study. The model was validated with less than 5% error. The implementation results show a reading latency around 500 clock cycles, at a very cost (only 12 monitored signal) regarding the accelerator area.

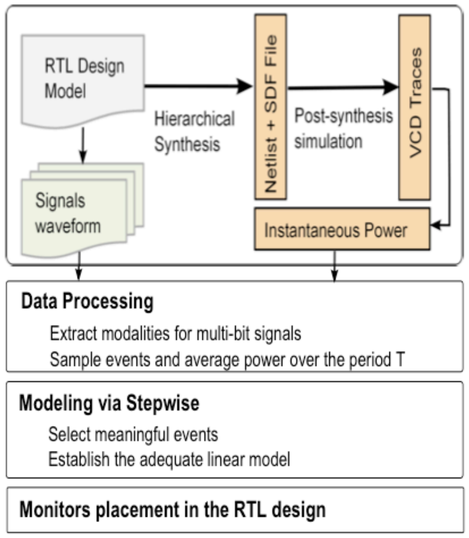

RTL Design Model
Hierarchical Synthesis
Netlist + SDF File
Post-synthesis simulation
VCD Traces
Signals waveform
Instantaneous Power
Data Processing
Extract modalities for multi-bit signals
Sample events and average power over the period T
Modeling via Stepwise
Select meaningful events
Establish the adequate linear model
Monitors placement in the RTL design

Fig. 11. Events profiling methodology flow

4.3 Software monitors and distributed database

Another complimentary approach to observe the system behavior is to use software monitors. As reported in section II, monitoring services are implemented into the Open-Scale RTOS to measure performance at the software level. The software threads implemented into the microkernel allow measuring the processor load, the communication load, application throughput, etc.

Applications suited to be run on Open-Scale are based on the traditional KPN (Kahn Process Network) paradigm. The application can be described as a finite set of tasks interconnected by software FIFOs. A given task can either be performed in hardware (with a coprocessor) or in software, in which case it becomes a thread executed by a processor on-chip. The performance of the system is directly related to the quality of the application mapping, *i.e.* the placement of threads on processors and the placement of software FIFOs on on-chip memories.

The processor load is simply calculated as the ratio between the number of cycles the PE is executing a set of threads, and the total number of cycles on a given time period. Indeed, due to local and remote thread dependencies, thread priorities, data availability, NOC traffic, etc., the processor may be more or less idled during the considered time period. This kind of information is very important to evaluate the efficiency of the application mapping for instance. For the same reasons, the software FIFOs may be filled more or less during application execution. Consequently, a specific service is implemented to measure the filling rate of the FIFOs.

It is mandatory for self-adaptive MPSOC architectures to be as reactive as possible to critical events, and to keep an accurate vision of the system behavior. A memory containing collected values from the different sensors is therefore needed, as well as appropriate and low cost means to store, handle, and retrieve these accumulated data. For this purpose, we have developed an in-memory database engine that fulfills these needs, as well as its associated API.

The DRET (Distributed Raw Events Table) is a distributed in memory database that is physically located in a specific part of the RAM memory of each NPU. Its purpose is to contain the monitoring data extracted from both hardware and software. The DRET of a tile may contain several tables like traditional database systems and each table contains several formatted events retrieved from the HW/SW sensors. The database has been designed to be an efficient data structure that can be used to store and retrieve information easily from the monitoring process. The used structure and the API are designed to keep the memory footprint and performance overhead as small as possible. The DRET can be seen as a uniform repository for the events whatever their origin (hardware sensor, software probe, local or external to a given processing element). While a DRET is mainly related to a given NPU, it can also be used to store a synthetic view of the state of neighboring NPU. In order to prevent the DRET from occupying too much memory space, a maximum size is associated to each table created in a DRET. A DRET table can also be cyclic. In that case, it behaves like a fixed-size cyclic buffer: inserting a new event in a full table discards the oldest one. The DRET and its API were evaluated in [37]. Although there is a slight memory overhead in the system imposed by the usage of the DRET, the overall gain can be very beneficial for the system. It allows the system to handle some decisions through the analysis and diagnosis of such stored events. This brings to the system the possibility to better decide the application mapping tuning it according to the requirements, and thus to reach the objective of chip self- adaptability.

5 Distributed Controllers

Prior works addressing system optimization (performance, power consumption, temperature, etc.) are generally based on centralized schemes. Application mapping, task placement and scheduling, voltage and frequency are tuned to adapt the design dynamic settings to its runtime constraints, according to a solution provided by a centralized algorithm. For instance, authors of [38] present a method to select the frequencies and voltages based on non-linear Lagrange optimization. The work presented in [39] proposes a centralized system inspired by Kirchhoff's current law to decide on the optimal power/performance configurations. Conventional centralized approaches become problematic when the number of cores increases. First, the complexity of the optimization problem explodes as the dimension of the system increases. Secondly, communication latencies induced after collecting on-line information have to be considered; in [40], authors use a time-stamped monitoring mechanism to ensure the reliability of their system, but this solution leads to overload the network traffic for designs connecting more than a few units. Some recent proposals focus on distributed techniques to optimize embedded systems subject to online information. In [41], the problem of multi-core systems thermal control is posed as a convex optimization problem and solved in a distributed manner using dual decomposition technique. Stochastic methods are introduced in [42] to manage power consumption over the chip lifetime. The complexity of the used Markov model increases with the number of possible power/performance configurations, making this technique inconvenient for designs embedding hundreds of cores.

In this work, we study a new approach in order to minimize the energy consumption of MPSOC at run-time taking into account different application constraints. As depicted in figure 12, we assume that each core is able to monitor its resources as described in the previous section (*i.e.* sensors to measure the system performance, and energy consumption), and a DVFS engine to tune the Voltage/Frequency couples. We propose to achieve Power/Performance tradeoff by distributed schemes based on 3 different approaches: PID based controllers, Game Theory inspired controllers, and Consensus Theory inspired controllers.

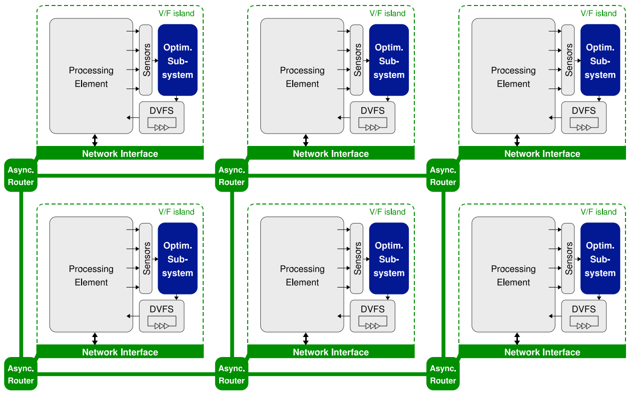


Fig. 12. Distributed Control overview

5.1 PID based Controllers

Due to the dynamic variations in the workload of MPSOC and its impact on energy consumption, adaptation techniques such as PID (Proportional-Integral-Derivative)-based control have been used to dynamically scale the voltage and the frequency of processors [43][44], and recently, of networks- on-chip [45][46]. These techniques differ in terms of adopted control parameters (*e.g.* task deadline, temperature) and response times (period necessary to stabilize a new voltage/frequency).

In [47], we have presented the following contributions: (i) power and energy consumption considered when tuning processor frequency; (ii) a PI-only controller proposed and compared to a PID-controller; and (iii) three perturbation scenarios with different application performance impact factors. The figure 13 illustrates an overview of the proposed approach. As it can be observed, one PID controller is devoted to each task in the system that must ensure soft-real time constraints. In this example, there is one task per NPU, so one PID controller for each processor is required. In the case where multiple tasks are executed in the same NPU, a system with multiple PID controllers in the same NPU could be proposed. The PID controller is implemented as a service in the Open-Scale RTOS. It only represents an overhead of less than 1% in terms of total memory required by the OS.

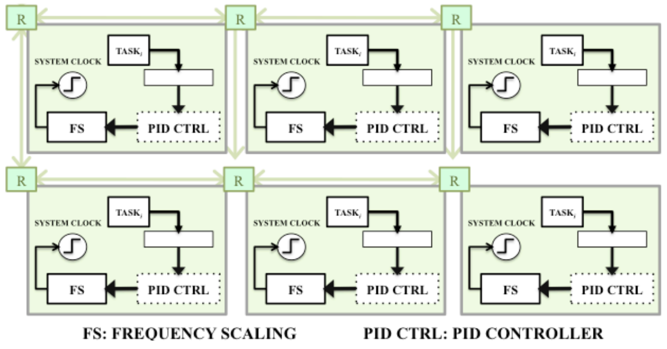


Fig. 13. MPSOC platform with PID controller

Monitoring information such as application throughput is fed into the PID controller module. It will match the actual throughput with the desired throughput (*setpoint*) and will then calculate an error value (*e*). This *e* value is used for calculating P, I and D parameters and as result, the PID controller will indicate a frequency in which the processor can reach a given *setpoint* according to reactiveness factor initially set. It is important to observe that the PID management and calculation are performed dynamically at run-time. In cases where current throughput is lower than the *setpoint,* the PID controller will select a frequency greater than the current one in order to reach the *setpoint* and to satisfy application performance constraints. On the other hand, when the current throughput is much higher than the expected one, the controller will sign to a lower frequency in order to reduce the power consumed by the NPU.

The proposed strategy consists in deciding controller parameters on a task basis. To this purpose, a SystemC/TLM-based Open-Scale simulation is executed in order to obtain the step-response. In this scenario, processor frequency is changed from 55MHz to 1005MHz and application throughput is monitored. The system is linear, that means the system's behavior remains the same under such frequency changes. Based on the high-level model, a number of different configurations of controllers can be explored. Each one exhibits different features such as speed, overshoot and static error. Once the process is modeled, PID parameters are fine tuned by using Simulink and the values of P, I and D are fed as input to the Open-Scale platform.

The PID strategy does not rely only on preventing application deadline-misses, but it also attempts to save energy, once the processor frequency is adjusted at real-time according to application requirements. Our approach can be easily integrated to linear systems and, platform resources utilization can therefore be optimized. By using PID controllers, we have shown in our simulation scenario that it was possible to save up to 32%, in terms of energy by tuning processor frequency according to application needs. As it can be noticed in [47], PID-controllers are intended to react faster under disturbing conditions compared to PI-only controllers. However its power consumption is higher. The good choice of which controller to use in multiprocessor system-on-chip platforms will be a trade-off between power consumption and the desired system's reactiveness.

5.2 Game-Theory based Controllers

Game theory involves a set of mathematical tools that describe interactions among rational agents. The basic hypothesis is that agents pursue well-defined objectives and take their knowledge and behaviors of other agents in the system into account to make their choices. In other words, it describes interactions of players in competitive games. Players are said to be rational since they always try to improve their score or advance in the game by making the best move or action. Game theory is based on a distributed model: players are considered as individual decision makers. For these reasons, game theory provides a promising set of tools to model distributed optimization on MPSOC and, moreover, this is an original approach in this context.

The second dynamic optimization proposed for Open-Scale is therefore inspired by game theory, where a *non-cooperative game* is a scenario with several players interacting by actions and consequences [48]. Basically, players individually choose an action within a defined set, resulting in consequences. Each player tries to maximize its outcome according to its preferences, leading to global optimization. If this sequence is repeated, under certain conditions, the game finds a solution formalized as *Nash Equilibrium*. These principles provide strong concepts to model the behavior of reactive systems where decisions are taken in a distributed and dynamic way, justifying the choice of the game theory for our approach.

We model the NPUs as players, the application latency and power consumption as a local objective function that depends on the global state of other NPUs. Then, a distributed algorithm selects the best solution. The objective functions are built by using different terms: the energy contribution of each PE to the whole energy consumption, the applicative latency contribution to the total latency and penalty functions modeling energy and latency constraints. We consider a MPSOC, composed of *n* NPUs interconnected by an asynchronous Network-on-Chip, such as Open-Scale. Each NPU integrates a DVFS engine that regulates the local voltage and frequency couple among a finite number of solutions. We denote T_i the clock period corresponding to NPU_i and $Ti- = (T_1,...,T_{i-1},T_{i+1},...,T_n)$ the periods of all other NPUs in the MPSOC. We assume that an external mechanism has mapped the application on the MPSOC, each NPU handling a unique task. The task assigned to NPU_i takes N_i cycles of T_i to be processed. We denote as L_{max} the application latency constraint and we consider that each task is scheduled every T_0 seconds. Energy, latency contributions and energy and latency constraints are modeled with this formalism. Then, two objective functions are built according to the formulated scenarios, *i.e.* energy or latency minimization under conditions.

The proposed method, based on Game Theory, optimizes the system while fulfilling dynamic constraints. A telecom test-case has been studied in [49,50] to demonstrate the effectiveness of this approach. For the evaluated case, the proposed technique has obtained up to 20% of latency gain under energy constraints, and 40% of energy gain under latency constraints.

Our studies have also shown in [51,52] that our method scales with the number of processors without excessive convergence times. For a 100-processor platform, our technique has required an average of 20 game cycles to reach the solution. A game cycle requires around 2500 cycles to collect monitoring data from other NPUs, minimize the objective function, and transmit its data to other NPUs [53]. The few calculation cycles needed to converge make this technique a feasible approach to optimize consumption and performance at run time. Furthermore, we have demonstrated that the achieved optimization is about 89% in average compared to a global offline method, which proves the quality of the results obtained with this method.

5.3 Consensus Theory based Controllers

Although the Game Theory approach is easy to implement, it may lead sometimes to local unstable minima, which necessitates extra resources to detect oscillations in the algorithm execution. Besides, application constraints are modeled as a penalty function, so real-time deadlines are not always guaranteed. Finally, global information must be shared between many NPUs, which could lead to undesired traffic in the interconnection network. To overcome these limitations, we attempt to apply gradient methods with consensus concepts in order to implement a cooperative and dynamic approach for Open-Scale.

Consensus is derived from the research on cooperative control theory. It was developed mainly for data processing in sensor networks and for multi-agents coordination. Consensus is defined as an iterative process utilizing a predefined message-passing protocol, leading a set of communicating elements to an agreement on a value or a common behavior [54]. Similarly to [55], we intend using the consensus to reach an agreement on one state optimizing a global interest in networked system.

A mathematical framework has been proposed for distributed optimization using hybrid approaches. This framework is a combination of subgradient methods with the consensus formalism to handle distributed optimization. In this section we summarize the key aspects of this theoretical framework before reporting afterwards its benefits.

In the context of consensus/subgradient optimization, each NPU is considered as an agent. The interconnection graph G is built with respect to the NPUs' NOC connections, and the state vector of the system is proportional to operating frequencies in each PE. Distributed models and algorithms such as Consensus are naturally adapted to Open-Scale.

Within a distributed cooperative scheme, each unit adjusts its local frequency, so that power consumption of the whole system is reduced without degrading performance. Considering our benchmark application, the proposed technique provides up to 45% energy gain under latency constraint changes, and up to 80% when different standards are applied. Our experiments have also shown in [56] that our distributed model is scalable, and can handle energy efficiency in future many core platforms; the number of communicating units can be sized to increase convergence speed and optimization quality. Indeed, for a 100-processor platform, our technique has required an average of 270 consensus cycles to reach the solution. One consensus cycle requires 500 cycles to collect monitoring data from other NPUs, compute an iteration of the consensus, and transmit its data to neighboring NPUs. We have estimated that the achieved optimization is about 82% in average compared to a global offline method.

6 Conclusion and Research Perspectives

After having exposed our vision of future self-adaptive embedded systems, we have presented our open-source MPSOC called Open-Scale. It is based on a building-block, a Network Processing Unit, mainly composed of a RISC processor, its local memory, peripherals, a network interface, and a set local sensors and actuators. The Open-Scale RTOS provides classical scheduling services, task, memory and interrupt management, but also a Message Passing Interface and dedicated services to support self-adaptability. This distributed prototyping platform has been used to investigate self-adaptation mechanisms. We have reported our researches in the design of distributed monitors, PVT sensors in FPGAs, activity counters, software monitors and a distributed embedded database to collect monitored data and its API. High-level distributed control approaches based on PID, Game Theory and Consensus Theory have been implemented and simulated. Our experimental results have shown that they are able provide 40% energy savings in average under application performance constraints (throughput or latency) in a distributed manner, at run-time. The induced overhead is low and scales well thanks, to their inherent distributed nature.

There are still several research challenges to reach the goal of self-adaptability. First, an intelligent management of monitored data from hardware and software is necessary, in order to correlate and select the most relevant approaches. The fine-tuning of mixed software and hardware actuators is a second research topic, since there are many ways to adapt the system to strike the balance between performance and energy consumption. Finally, the learning capabilities of such distributed systems must be explored, in order to achieve certainly in a near future, real autonomous chips.

7 Acknowledgement

This work was supported in part by the French National Research Agency (ANR), in collaboration with CEA LETI and LIP6. The author gratefully acknowledges the contributions of many people, who helped in different ways to complete this work, the LIRMM colleagues, Lionel Torres, Gilles Sassatelli, Michel Robert, Luciano Ost, current and former PhD candidates, especially Gabriel Marchesan Almeida, Diego Puschini, Imen Mansouri, Nicolas Hébert, Nicolas Saint-Jean, Florent Bruguier, Lyonel Barthe and Rémi Busseuil.

8 References

[1] The International Technology Roadmap for Semiconductors, System Drivers Chapter, 2011, [online] http://www.itrs.net/Links/2011ITRS/2011Chapters/2011SysDrivers.pdf

[2] S. Borkar, T. Karnik, S. Narendra, J. Tschanz, A. Keshavarzi, and V. De, *"Parameter variations and impact on circuits and microarchitecture",* Design Automation Conference, 2003. Proceedings*,* pp. 338–342.

[3] O. Unsal, J. Tschanz, K. Bowman, V. De, X. Vera, A. Gonzalez, and O. Ergin, *"Impact of Parameter Variations on Circuits and Microarchitecture"*, *IEEE Micro,* vol. 26, no. 6, pp. 30–39, 2006.

[4] N. Saint-Jean, G. Sassatelli, P. Benoit, L. Torres, and M. Robert, *"HS-Scale: a Hardware-Software Scalable MP-SOC Architecture for embedded Systems"*, IEEE Computer Society Annual Symposium on VLSI (ISVLSI '07), IEEE, 2007, pp. 21-28

[5] P. Guerrier and A. Greiner, *"A generic architecture for on-chip packet-switched interconnections"*, in DATE '00: Proceedings of the 2000 Design, Automation and Test in Europe Conference and Exhibition, pages 250–256, 2000.

[6] William J. Dally and Brian Towles, *"Route packets, not wires: on-chip inteconnection networks"*, In DAC '01: Proceedings of the 38th conference on Design automation, pages 684–689, New York, NY, USA, 2001. ACM.

[7] L. Benini and G. De Micheli, *"Networks on chips: a new SoC paradigm"*, IEEE Computer, 35(1):70–78, Jan 2002.

[8] Tobias Bjerregaard and Shankar Mahadevan, *"A survey of research and practices of Network-on-chip"*, ACM Computing Surev, 38(1):1, 2006.

[9] Partha Pratim Pande, C. Grecu, M. Jones, A. Ivanov, and R. Saleh, *"Performance evaluation and design trade-offs for network-on-chip interconnect architectures"*, Computers, IEEE Transactions on, 54(8):1025–1040, Aug. 2005.

[10] D. Bertozzi and L. Benini, *"Xpipes: a network-on-chip architecture for gigascale systems-on-chip"*, Circuits and Systems Magazine, IEEE, 4(2):18–31, 2004.

[11] E. Beigne, F. Clermidy, P. Vivet, A. Clouard, and M. Renaudin, *"An Asynchronous NOC Architecture Providing Low Latency Service and Its Multi-Level Design Framework"*, In ASYNC '05: Proceedings of the 11th IEEE International Symposium on Asynchronous Circuits and Systems, pages 54–63, Washington, DC, USA, 2005. IEEE Computer Society.

[12] J. Pontes, M. Moreira, R. Soares, and N. Calazans. *"Hermes-glp: A gals network on chip router with power control techniques"*, In Symposium on VLSI, 2008. ISVLSI'08. IEEE Computer Society Annual, pages 347–352, April 2008.

[13] Umit Y. Ogras, Radu Marculescu, Puru Choudhary, and Diana Marculescu, *"Voltage-frequency island partitioning for GALS-based Networks-on-Chip"*, In DAC '07: Proceedings of the 44th Annual Conference on Design Automation, pages 110–115, New York, NY, USA, 2007. ACM.

[14] James Donald and Margaret Martonosi, *"Techniques for multicore thermal management: Classification and new exploration"*, In ISCA '06: Proceeding of the 33rd International Symposium on Computer Architecture, pages 78–88, 2006.

[15] Edith Beigne, Fabien Clermidy, Sylvain Miermont, and Pascal Vivet, *"Dynamic voltage and frequency scaling architecture for units integration within a gals NOC*", In NOCS, pages 129–138, 2008.

[16] Edith Beigne, Fabien Clermidy, Sylvain Miermont, Alexandre Valentian, Pascal Vivet, S Barasinski, F Blisson, N Kohli, and S Kumar, *"A fully integrated power supply unit for fine grain DVFS and leakage control validated on low-voltage SRAMs"*, In ESSCIRC'08: Proceeding of the 34th European Solid-State Circuits Conference, Edinburg, UK, Sept. 2008.

[17] Barthe L., Cargnini L. V., Benoit P., Torres L., *"Optimizing an Open-Source Processor for FPGAs: A Case Study"*, IEEE FPL'11: Field Programmable Logic and Applications (2011), Greece, pp. 551-556

[18] L. Barthe, L. V. Cargnini, P. Benoit and L. Torres, *"The SecretBlaze: A Configurable and Cost-Effective Open-Source Soft-Core Processor"*, 25th IEEE International Parallel & Distributed Processing Symposium, May 16-20, 2011, Anchorage (Alaska) USA, pp. 310-313

[19] F. Moraes, N. Calazans, A. Mello, L. Moller, and L. Ost, *"Hermes: an infrastructure for low area overhead packet- switching networks on chip"*, Integration VLSI Journal, vol. 38(1), 2004, pp. 69–93.

[20] O. Richard Herveille, *"Wishbone System-on-Chip (SoC) Interconnection Architecture for Portable IP Cores"*, Revision B.4, OpenCores, 2010. [Online]. Available at: http://www. opencores.org/

[21] S. Rhoads, *"Plasma - most mips i(tm)"* [Online]. Available at: http://www.opencores.org/project,plasma

[22] G. Kahn and D.B. MacQueen, *"Coroutines and networks of parallel programming"*, In B. Gilchrist, editor, Information Processing 77: Proceedings of the IFIP Congress 77, Toronto, Canada, August 8-12, 1977, pages 993–998. North-Holland, 1977.

[23] Busseuil R., Barthe L., Almeida G. M., Ost L., Bruguier F., Sassatelli G., Benoit P., Robert M., Torres L., *"Open-Scale: A Scalable, Open-Source NOC-based MPSoC for Design Space Exploration"*, IEEE International Conference on Reconfigurable Computing and FPGAs (ReConFig), 2011, pp. 357-362

[24] M. Nourani, A. Radhakrishnan, *"Testing On-Die Process Variation in Nanometer VLSI"*, IEEE Design & Test of Computers, Volume 23, Issue 6, June 2006 Page(s):438 – 451

[25] SB Samaan, *"Parameter Variation Probing Technique"* US Patent 6535013, 2003

[26] M. Persun *"Method and apparatus for measuring relative, within-die leakage current and/or providing a*

temperature variation profile using a leakage inverter and ring oscillators" US Patent 7193427 (2007)

[27] H-J Lee, *"Semiconductor device with speed binning test circuit and test method thereof "* US Patent 7260754

[28] Z. Abuhamdeh, B. Hannagan, Jeff Remmers, Alfred L. Crouch *"A Production IR-Drop Screen on a Chip"*, IEEE Design & Test of Computers, Volume: 24, Issue: 3, pp. 216-224, 2007

[29] A. Drake et al. *"A Distributed Critical Path Timing Monitor for A 65nm High Performance Microprocessor",* ISSCC 2007, pp.398-399.

[30] S. Lopez-Buedo, J. Garrido, and E. Boemo, *"Dynamically inserting, operating, and eliminating thermal sensors of FPGA- based systems",* IEEE Transactions on Components and Packaging Technologies, vol. 25, no. 4, pp. 561–566, 2002.

[31] A. Drake, *"Adaptive Techniques for Dynamic Processor Optimization",* Series on Integrated Circuits and Systems. Boston, MA: Springer US, 2008.

[32] Bruguier F., Benoit P., Torres L., *"Investigation of Digital Sensors for Variability Characterization on FPGAs"*, ReCoSoC'10: 5th International Workshop on Reconfigurable Communication-Centric Systems on Chip, France, pp. 95-100

[33] Ganesan, K.; John, L.; Salapura, V.; Sexton, J.; , *"A Performance Counter Based Workload Characterization on Blue Gene/P",* Parallel Processing, 2008. ICPP '08. 37th International Conference on , vol., no., pp.330-337, 9-12 Sept. 2008

[34] J.M.May, *"MPX: Software for multiplexing hardware performance counters in multithreaded programs",* in Parallel and Distributed Processing Symposium, Proceedings 15th International, p. 8, April 2001

[35] K. Jihong and K. Yongmin *" Performance Analysis and Tuning for a Single-Chip Multiprocessor DSP",* IEEE Parallel Distrib. Technol. , vol.5 , pp. 68-79, Jan 1997 . Los Alamitos, CA, USA.

[36] K. Hyun-min et al. *" Performance monitor unit design for an AXI-based multi-core SoC platform",* Proceedings of the 2007 ACM symposium on Applied computing, SAC '07,pp. 1565-1572, Seoul, Korea.

[37] E. Faure, G.M. Almeida, M. Benabdenbi, P. Benoit, F. Clermidy, F. Pêcheux, G. Sassatelli, L. Torres, *"An in-memory monitoring database for self adaptive MP2SoCs"*, Design and Architectures for Signal and Image Processing (DASIP), 2010 Conference on, 2010, pp. 97 – 104

[38] Niyogi, K. and Marculescu. *"Speed and voltage selection for GALS systems based on voltage/frequency islands",* ASP-DAC '05. ACM, New York, 292-297

[39] Deniz, Z.T.; Leblebici, Y.; Vittoz, E.A., *"On-Line Global Energy Optimization in Multi-Core Systems Using Principles of Analog Computation",* Solid-State Circuits, IEEE Journal of vol.42, no.7, pp.1593-1606, July 2007

[40] S. Madduri, et al., *"A monitor interconnect and support subsystem for multicore processors",* in the Proc. of the IEEE/ACM Design Automation and Test in Europe Conference, Nice France, pp. 761-766, 2009

[41] Mutapcic, A. et al., *"Processor speed control with thermal constraints".* Trans. Cir. Sys. Part I 56, 9 (Sep. 2009), 1994-2008.

[42] H. Jung and M. Pedram, *"Uncertainty-Aware Dynamic Power Management in Partially Observable Domains",* IEEE Trans. on VLSI Systems, 2009.

[43] Q.Wu,P.Juang, *et al.* , *"Formal online methods for voltage/frequency control in multiple clock domain microprocessors",* SIGARCH Comput. Archit. News*,* vol. 32, pp. 248–259, October 2004

[44] Y. Zhu and F. Mueller, *"Feedback EDF scheduling exploiting hardware- assisted asynchronous dynamic voltage scaling",* SIGPLAN Not., vol. 40, pp. 203–212, June 2005

[45] U. Y. Ogras, R. Marculescu, and et al., *"Variation-adaptive feedback control for networks-on-chip with multiple clock domains",* Proceedings of the 45th annual Design Automation Conference (DAC'08)*,* pp. 614–619, June 2008

[46] A. Sharifi, H. Zhao, and et al., *"Feedback control for providing qos in noc based multicores",* Proceedings of the Conference on Design, Automation and Test in Europe (DATE'10), pp. 1384–1389, March 2010

[47] G. Almeida, R. Busseuil, L. Ost, F. Bruguier, G. Sassatelli, P. Benoit, L. Torres, M. Robert, *"PI and PID Regulation Approaches for Performance-Constrained Adaptive Multiprocessor System-on-Chip"*, Embedded Systems Letters, IEEE, Volume: PP, Issue:99, ISSN: 1943-0663, DOI: 10.1109/LES.2011.2166373, September 2011, pp. 1-4

[48] M.J. Osborne and A. Rubinstein, *"A Course in Game Theory"*. MIT Press, 1994

[49] D. Puschini, F. Clermidy, P. Benoit, G. Sassatelli, and L. Torres, *"Adaptive energy-aware latency-constrained DVFS policy for MPSoC"*, 2009 IEEE International SOC Conference (SOCC), IEEE, 2009, pp. 89-92

[50] D. Puschini, F. Clermidy, P. Benoit, G. Sassatelli, and L. Torres, *"Dynamic and distributed frequency assignment for energy and latency constrained MP-SoC"*, Design, Automation & Test in Europe Conference & Exhibition, 2009. DATE '09., 2009, pp. 1564-1567

[51] D. Puschini, P. Benoit, F. Clermidy, G. Sassatelli, *"A Game-Theoretic Approach for Run-Time Distributed Optimization on MP-SoC"*, International Journal of Reconfigurable Computing, Volume 2008, 403086 (2008), pp. 1-10

[52] D. Puschini, F. Clermidy, P. Benoit, G. Sassatelli, and L. Torres, *"Temperature-Aware Distributed Run-Time Optimization on MP-SoC Using Game Theory"*, Symposium on VLSI, 2008. ISVLSI '08. IEEE Computer Society Annual, 2008, pp. 375-380

[53] I. Mansouri, P. Benoit, D. Puschini, L. Torres, F. Clermidy, G. Sassatelli, *"Dynamic Energy Optimization in NoC-based System-on-Chips"*, Journal of Low Power Electronics JOLPE – Vol. 6, N° 4, December 2010, pp. 564-577

[54] Olfati-Saber, J. A. Fax, and R. M. Murray. *"Consensus and cooperation in networked multi-agent systems,"* Proceedings of the IEEE, 95(1): 215233 (2007).

[55] Johansson, B. and al.,"Subgradient *methods and consensus algorithms for solving convex optimization problems".* Decision and Control, CDC , Dec. 2008

[56] Mansouri I., Clermidy F., Benoit P., Torres L., *"A Run-time Distributed Cooperative Approach to Optimize Power Consumption in MPSoCs"*, SOCC'10: 23th IEEE International SOC Conference, United States, pp. 25-30