Hierarchical Energy Monitoring for Many-core Systems

André L. M. Martins[1], Marcelo Ruaro[1,2], Fernando G. Moraes[1]
[1]FACIN - PUCRS – Av. Ipiranga 6681, 90619-900, Porto Alegre, Brazil
[2]SETREM, Computer Science Department, Três de Maio, Brazil
marceloruaro@setrem.com.br, andre.del@acad.pucrs.br, fernando.moraes@pucrs.br

***Abstract*—Power/energy managers for many-core systems has been proposed as a possible solution for the restricted power constraints imposed by new chip technologies. Therefore, when these management approaches are implemented on real systems, strong limitations to apply the management techniques has been reported. The hierarchical energy monitoring method presented in this paper provides a multi-level energy data for power/energy management. The proposed energy monitoring is implemented at the software level on a real homogeneous NoC-based many-core platform, with low intrusiveness on hardware. The energy monitoring data is transmitted through packets on the network with low impact on system performance.**

Keywords—NoC; many-core; energy monitoring

I. INTRODUCTION

The *utilization wall*, known as Dark Silicon [1], consists of maintaining some percentage of the chip area powered off (*dark*) to respect some power and/or thermal constraint. In new technologies, more significant is the leakage power, and more silicon area should be dark. Managing many-cores under this new paradigm brings new design challenges and opportunities.

Many-cores management requires accurate information (e.g. power, energy, temperature) to control the system under restricted power budget imposed by the utilization wall. Thus, a monitoring scheme should provide power data to a controller, allowing the many-core management. Therefore, a *monitoring* scheme is key for the effectiveness of the controller on real systems.

Haghbayan et al. [2] propose a feedback control approach for power management on NoC-based many-cores running dynamic workloads. The model assumes that each router has a congestion monitor and a power sensor. The power sensor generates monitoring data stored in matrices. The congestion data defines the applications mapping, and the controller sets new values for voltage and frequency respecting TDP (Thermal Design Power) constraints.

Yu et al. [3] present a thermal management for heterogeneous MPSoCs running adaptive workloads. The thermal model is adapted from HotSpot. Considering the behavior of the target workloads, the monitor determines the state of the task as "hot" or "cool". If the task is "hot", the frequency scaling algorithm may decrease the task frequency. The equivalent policy is made for "cool" tasks.

Bogdan et al. [4] present an optimal control approach for power management on NoC-based many-cores. A fractional calculus is used to model real workloads on a fractal controller. The Authors show that the controller achieves optimal results in power savings applying mapping and DVFS techniques, as well as the synthesis of the power manager on FPGA. Besides that, according to the Authors, the controller is not scalable, and the benchmark used on experiments is restricted to only one application.

Hanumaiah et al. [5] propose an energy-efficient manager for heterogeneous many-cores. Their framework supports DVFS, task migration, and fan speed control to determine the optimal configuration to maximize the energy efficiency. The model considers the monitoring sampling and actuation timing. A quad-core Intel Sandy Bridge processor runs the proposed controller. However, on a real processor, only a limited version of the DVFS technique is applied because the system provides one temperature per core and overall system power. Besides, the sampling rate of sensors is lower than expected by the model and noisy.

Muthukaruppan et al. [6] present a power management framework integrated within the Linux on an ARM big.LITTLE platform. The board used for experiments contains frequency, voltage, power and energy sensors per cluster of processors and allows setting of voltage and frequency dynamically per cluster. Besides that, the manager receives data periodically about the performance of applications. The actuation policies supported are DVFS per cluster and task migration per core and per cluster keeping the tasks under their QoS rates.

Some proposals present new perspectives on many-core management considering the utilization wall issue [7]. Raghunathan et al. [8] explore process variation between cores of a homogeneous many-core to choose the most energy-efficient set of cores to a given application. The proposed manager is highly dependent on the variability of the chip. With a small variability, the manager does not improve the performance.

Kriebel et al. [9] present an adaptive soft error resilience manager. At design-time, a reliability score is attributed to a set of applications, and a library of cores that have the same ISA containing different reliability features is generated. After, a tool generates a many-core with cores of this library. At run-time, the proposed manager uses reliability scores of applications to map them on adequate cores, according to a level of protection required by the applications while respecting area and power constraints.

A gap in the literature is the monitoring approach. In [8] and [9] it is not possible to identify how the power/energy data is obtained and computed at runtime. In [2]–[4] it is assumed on the model that the power data is ready to be used, but the power data can be available in a different scheme on real systems then expected by the model, as reported in [5], limiting the management or adding steps of remodeling the model's parameters. The assumption that power or energy data is available on these high-level models abstracts the complexity of monitoring in real systems. The sampling of the monitored data and the time required for actuation to take effect should be considered on the controller [5], to keep the real behavior close to expected by the model. The system manager applies a given actuation policy when the energy/power monitoring data is available, but this information (when available) is performed at system level [5] or restricted at the cluster level [6]. TDP is the global reference for maximum power on systems, normally used for the Dark Silicon-aware managers [8]–[10] as an upper bound. Thus, mainly for the models with power sensor per core, the system must adopt a scheme to count the overall power/energy to compare to TDP, as well, the design must model the generation and the transmission of power/energy data in predicted level.

This work presents a hierarchical monitoring method to allow the energy monitoring on power management systems. The method has the following features: (*i*) temperature/power sensors are not required; (*ii*) configurable sampling data; (*iii*) multi-grained data, i.e., the volume of the collect date is a function of the hierarchy level; (*iv*) implemented at the software level, with a small hardware intrusiveness; (*v*) scalable. This monitoring scheme is integrated into a real many-core system, generating instant energy and accumulated energy values for large workloads on different many-core sizes.

II. MANY-CORE MANAGEMENT ARCHITECTURE

The current work assumes a many-core with distributed resource management [11] to enable the hierarchical monitoring method. Fig. 1 shows a simplified view of the many-core architecture. The hardware of each PE (Processing Element) is the same. Each PE contains at least one processor, DMA, private memory, NI (network interface) and the NoC router. The many-core is divided into virtual regions, named *clusters*.

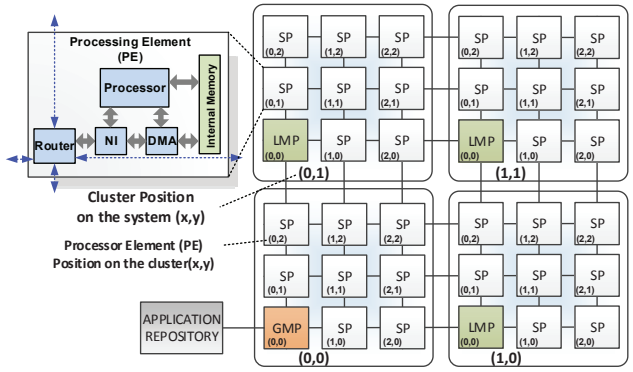

Fig. 1. 6x6 NoC-based many-core instance, with four 3x3 clusters.

The software running at each PE defines its role. Each PE may assume one of three roles:

- SP, slave processor, contains a multi-task OS (operating system) and executes user tasks. Inter-task communication occurs through message-passing (MPI-like API);
- LMP, local manager processor, responsible for managing the cluster, executing actions as task mapping, task migration, QoS control, communication with other manager processors;
- GMP, global manager processor, executes all tasks of the LMP and global management functions. Example of such functions includes: initialization of the clusters, selection of a cluster to execute a given application, access to external devices (as a memory) to transmit the object code of the tasks to the select SP.

The GMP knows the state of each cluster (e.g. number of available resources), and the LMP has a detailed view of the cluster with accurate information related to the SPs it controls. Such hierarchical management ensures scalability since it is not necessary to have one PE managing all SPs of the system.

III. ENERGY MONITORING METHOD

The design of the energy monitoring method is hierarchical (Fig. 2), according to the many-core architecture previously presented. The proposed scheme incurs in low traffic in the NoC, with a small intrusiveness in the hardware and software of the PE.

The monitoring scheme is divided into three levels. The lowest level of the monitoring scheme is applied on SPs, which compute the spent energy, sending the *energy per processor* to the LMP periodically. On the next level, the LMP receives the energy data of its corresponding SPs to compute the energy of the cluster. Finally, the LMPs send to the GMP, on the top level of the hierarchy, the *energy per cluster*. The SPs of the GMP cluster send the *energy per processor* data directly to the GMP. The GMP (Fig. 2) can estimate the *overall system energy* because it receives energy data from all LMPs, and from the SPs it controls.

A. Energy Monitoring at Processor Element Level

Martins et al. in [12] present a method to estimate the power and the energy consumption through characterization of the processor by a calibration process. The instruction set is grouped into classes and for each instruction class, an RTL simulation of the corresponding assembly class code is executed on the netlist of the processor. The method presented a small error compared to real measurements (below 5%), providing a measure of *energy per instruction* for all classes of instructions. The proposed energy monitoring method uses this approach on PEs, but small adaptations at the hardware and software are required.

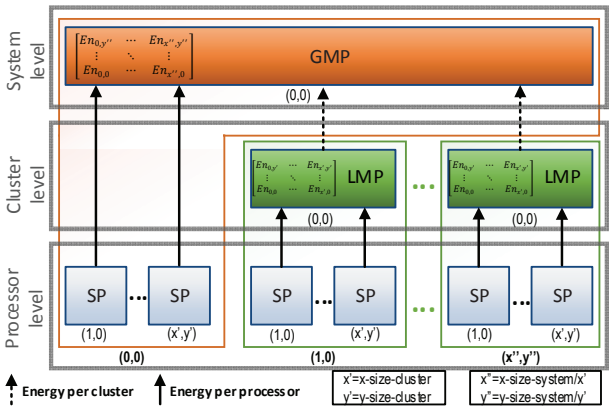

Fig. 2. Hierarchical energy monitoring scheme on distributed many-core.

Hardware requirements. Instruction counters are added to the processor hardware of the PE. In the current implementation, such counters are included in the control part of the processor. If the hardware of the processor cannot be modified, a sniffer may be added in the address and instruction buses. Specific purpose registers store the number of executed instructions per class. The instructions per class registers are continuously updated.

Software requirements. In the OS, new functions read the instruction counters periodically, according to a sampling window, to compute the energy spent by the processor. Multiplying the energy per instruction by the number of instructions, the OS may estimate the energy of the processor on such window. Average power, accumulated energy, temperature may be either estimated from the energy per instruction and time samplings. Next, the OS sends a monitoring packet to the LMP with the PE address, timestamp, and energy in the sampling window. The OS only sends the monitoring packets when there are tasks running on the PE.

Note that, the proposed method does not assume the existence of any sensors on the hardware, as most related works. Also, the required hardware is restricted to instruction counters, making this method low intrusive and easily adaptable to different processors. Related works that implement power management on real systems report error and noise on the sensors, making the proposed method an alternative or complement to these methods. Besides, energy data at the PE level is usually not available on real systems, even when a policy management is applied at PE level.

Small values for the sampling window induce a higher execution time overhead and higher traffic on the network. On the other hand, the power manager needs energy/power data periodically to manage the system adequately. Therefore, the configurable sampling window enables the exploration between execution overhead and management quality.

B. Energy Monitoring at Cluster Level

The LMP receives periodically packets with the *energy per processor* data from the SPs that are running tasks in the cluster. A matrix stores the *energy per processor* data of all LMPs of the cluster. The SP position on the cluster is the index of the *energy per processor* matrix (Fig. 1). The *energy per cluster* is obtained by summing the *energy per processor* values transmitted in these monitoring packets. Also, the timestamp of the packet is compared to the current time, enabling to evaluate the congestion of the monitoring scheme.

The LMPs send the energy per cluster data for the GMP

periodically, according to a second sampling window. Similarly to SPs, the LMP sends packets only when there is an application running in the cluster. The *energy per cluster* sampling window is the same for all clusters and independent from the *energy per processor* sampling window. The OS of LMPs runs the energy monitoring at the cluster level, and there is no need of hardware modifications on this level.

The *energy per cluster* enables a set of power management policies. Related works use some of these policies, applying voltage/frequency islands [2], [4] on a groups of PEs (clusters), and power gated the entire cluster [6]. Also, the selection of a cluster for mapping a new application, and the decisions related to task migration are examples of techniques that *energy per cluster* is useful.

C. Energy Monitoring at System Level

At the top level of the energy monitoring scheme, the GMP receives the *energy per cluster* from all LMPs and estimates the *overall system energy*, similarly to the *energy per cluster* computation, by summing the values received in all *energy per cluster* packets. The GMP stores every *energy per cluster* data in a matrix (Fig. 2). The index of the matrix indicates the position of the cluster in the system related to GMP, which is the cluster (0,0).

According to the many-core architecture, only the GMP has access to the application repository. Thus, the GMP has an overall view of the system, allowing it to decide which cluster an application will execute. Besides that, the *overall system energy* might be used for other management goals at the system level. For example, assuming that the TDP is the constraint for the management system ([2],[3],[6],[9],[10]), the energy monitor provides the *overall system energy* data to the manager, that can trigger an actuation policy whether necessary.

IV. RESULTS

The experiments are executed on a clock cycle-accurate SystemC RTL homogeneous many-core [11] running at 100MHz. Five applications modeled as task graphs (applications described in C language) execute in the experiments: MPEG (5 tasks), DTW (6 tasks), multispectral image analysis (14 tasks), synthetic (6 tasks), producer-consumer (2 tasks). Five distinct many-cores instances were simulated, varying the number of PEs, the cluster size, and the number of applications. One instance of each application starts at every 5 ms.

A. Monitoring Data

Fig. 3 presents energy data obtained by the hierarchical energy monitoring. The first row of graphs presents results related to the instant energy, i.e. the amount of energy computed according to the sampling window. The second row of graphs shows the accumulated energy, i. e. the sum of all past instant energy data.

The setup used on the graphs is a 4x4 many-core divided into four 2x2 clusters, running twelve applications. The x-axis corresponds to the number of clock cycles (tick counter). Monitoring sampling between SPs and LMP is set to 40,000 clock cycles, and 100,000 clock cycles between LMPs and GMP. This small scenario is presented for the sake of simplicity, generating readable plots.

The first column of the Fig. 3 presents energy data generated by an SP at the **processor level** (in fact 12 plots can be generated, one for each SP). The red vertical lines signalize when a task is allocated or terminated. The instant energy is computed only when there are tasks running. For example, at 1,500 Kticks there are no tasks running and, then, there is no instant energy sampling. Also, the instant energy graphs register different energy profiles of tasks. On the graph, for example, it is possible to observe tasks spending between 400nJ and 1200nJ. The accumulated energy graph represents the total energy spent in the PE. The value in the graph remains constant when there are no tasks. At the end of the simulation, it is possible to observe that the PE spent 75 uJ during the entire simulation.

The second column of the Fig. 3 shows the energy at the **cluster level**, for the cluster 1x0 (in fact four plots can be generated, one for each cluster). The instant energy graph gives an overall view of the energy spent by the SPs in the cluster. For example, two SPs of the cluster 1x0 have a high computation load while one processor has a small load. The accumulated energy graph shows a similar result. If a power manager policy were applied, a distinct behavior should be observed, with a uniform consumption in all SPs.

The **overall system level** is shown in the last column of the graph. By the curves provided by the GMP view, observe that three clusters running applications on this test case and only the cluster 0x0 do not receive applications. Moreover, the system takes around 5,000 Kticks (50 ms) to run all the tasks. Note that, when a cluster has no application to run, no energy data packet is sent. Also, details related to task scheduling (available at the processor level) or which SPs are currently running (available at the cluster level) are abstracted on the system level.

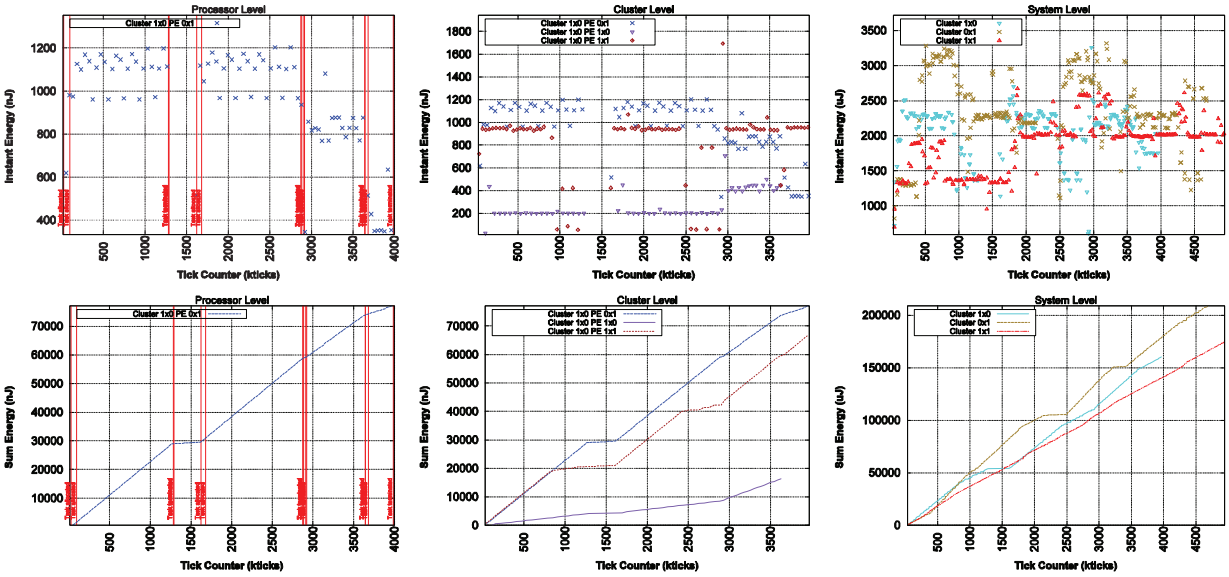

Fig. 3. The columns show different levels of instant energy (first row) and accumulated energy (second row) reported by proposed energy monitoring method.

TABLE I. Energy monitoring support overhead evaluation for different many-cores

Manycore size	Cluster size	N# app.	Reference exec. time	Execution time penalty (%) according to cluster/global sampling windows, in thousands of clock cycles (Kticks)					
				20 Kticks/ 30 Kticks	40 Kticks/ 100 Kticks	60 Kticks/ 150 Kticks	200 Kticks/ 300 Kticks	400 Kticks/ 1000 Kticks	600 Kticks/ 1500 Kticks
6x6	3x3	24	46.64 ms	13.72%	9.78%	8.15%	6.41%	6.41%	6.05%
8x8	4x4	30	49.32 ms	23.09%	21.05%	6.91%	11.66%	9.77%	0.81%
8x8	4x4	60	46.33 ms	16.90%	15.41%	15.13%	8.16%	8.94%	9.19%
9x9	3x3	36	81.02 ms	3.53%	12.95%	4.63%	7.05%	3.06%	5.86%
9x9	3x3	72	50.70 ms	4.28%	3.89%	4.02%	2.72%	3.47%	2.52%
			Average Overhead:	**10.46%**	**11.48%**	**6.67%**	**6.09%**	**5.19%**	**3.75%**

B. *Monitoring Overhead Evaluation*

Table I shows the performance overhead of the energy monitoring scheme for systems up to 81 PEs. The first three columns present the many core size, the cluster size, and the number of the applications executing in the test case. The 4th column contains the reference execution time, i.e., without the energy monitoring method. The next columns vary the sampling window of sending packets between SP and LMP (cluster level) and between LMP and GMP (system level). The time slice of the OS (time to schedule the tasks) is 20 Kticks, constraining the minimum sampling window to this value.

Results presented from Table I may be summarized as follows: (*i*) the greater the sample windows, smaller the impact on the performance, from 10.46% to 3.75% (expected result); (*ii*) larger clusters (4x4 compared to 3x3 for example) penalizes the execution time since more monitoring are generated. Therefore, results of the overhead of monitoring are related to the mapping of tasks/applications, which changes according to the sampling window. For this platform, it is suggested monitoring windows of 200K/300Kticks since these values represent a good tradeoff between monitoring events and performance penalty (6.09% in average).

The impact of the monitoring traffic must be also measured. The scenario used for the results plotted in Fig. 4 is the same in the previous section (4x4 many-core, 40K/100Kticks for the sampling windows). The *Dif Time* (y-axis) presents the time between the injection and the reception of each monitoring packet (the packet timestamp enables this measure). The x-axis is the simulation time. The labels at each point correspond to SP position in the cluster on the first graph, and the cluster position in the system on the second graph.

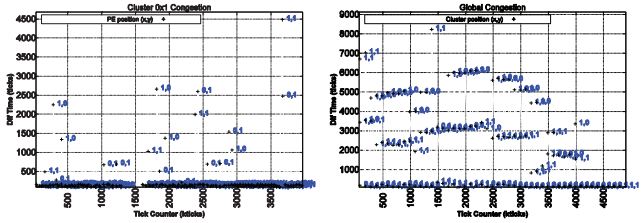

Fig. 4. Analysis of the NoC congestion generated by energy monitoring.

Most of the packets in Fig. 3 require 150 ticks to be transmitted (94.94%), but some packets may delay due to congestion. For the cluster 0x1, the worst case is around 4,500 tick (10% of cluster level sampling). Note that, as shown in Fig. 3, when there are no tasks running, no packets are sent. For the system level, 54.19% of packets get congestion effect, even with the large sampling window, but the worst case of congestion is below 10% (8,000 ticks for the cluster 1x1). Since the cluster/global sampling window is greater or equal than 40k/100kticks, every energy monitoring packet gets low congestion (10% of sampling window).

V. CONCLUSIONS AND FUTURE WORKS

This paper presented a hierarchical energy monitoring method for many-core systems. The method is applied on a real many-core platform with minimum hardware intrusiveness. The energy monitoring data is computed from the number of executed instructions and the energy per instruction previously calibrated at the physical level. The monitoring results showed that the hierarchical scheme provides distinct energy data grain at processor, cluster and system levels, while the settings of monitoring sampling window do not imply on hard congestion nor on high delay of arrival time of packets.

As future works, the design space of dynamic energy/power management for many-core systems will be deployed, based on the flexibility of the proposed hierarchical energy monitoring method. The instant energy data and the accumulated energy data can be used for triggering controls policies (like mapping/migration, power gating, etc.) of many-core managers, at run-time, under restricted power constraints while achieve different goals like reliability, and energy efficiency.

ACKNOWLEDGMENTS

The Author Fernando Moraes is supported by CNPq - projects 472126/2013-0 and 302625/2012-7, and FAPERGS - project 2242-2551/14-8.

REFERENCES

[1] H. Esmaeilzadeh; E. Blem; R. St. Amant; K. Sankaralingam; D. Burger. “Dark silicon and the end of multicore scaling”. IEEE Micro, vol. 32(3), 2012, pp. 122–134.

[2] M. Haghbayan; A. Rahmani; A. Y. Weldezion; P. Liljeberg; J. Plosila; A. Jantsch; H. Tenhunen. “Dark Silicon and Congestion Aware Power Management for Manycore Systems under Dynamic Workloads”. In: ICCD, 2014, pp. 509-512.

[3] H. Yu; R. Syed; Y. Ha. “Thermal-aware frequency scaling for adaptive workloads on heterogeneous MPSoCs”. In: DATE, 2014, 6p.

[4] P. . Bogdan; R. Marculescu; S. Jain. “Dynamic power management for multidomain system-on-chip platforms: An optimal control approach”. ACM TODAES, vol. 18(4), 2013, pp. 46–66.

[5] V. Hanumaiah; S. Vrudhula. “Energy-efficient operation of multicore processors by DVFS, task migration, and active cooling”. IEEE Trans. Comput., vol. 63(2), 2014, pp. 349–360.

[6] T. S. Muthukaruppan; M. Pricopi; V. Venkataramani; T. Mitra; S. Vishin. “Hierarchical power management for asymmetric multi-core in dark silicon era”. In: DAC, 2013, 9p.

[7] M. Shafique; S. Garg; J. Henkel; D. Marculescu. “The EDA Challenges in the Dark Silicon Era”. In: DAC, 2014, 6p.

[8] B. Raghunathan; Y. Turakhia; S. Garg; D. Marculescu. “Cherry-picking: Exploiting process variations in dark-silicon homogeneous chip multi-processors”. In: DATE, 2013, pp. 39–44.

[9] F. Kriebel; S. Rehman; D. Sun; M. Shafique; J. Henkel. “{ASER}: Adaptive Soft Error Resilience for Reliability-Heterogeneous Processors in the Dark Silicon Era”. In DAC, 2014, pp. 12:1–12:6.

[10] Y. Turakhia; B. Raghunathan; S. Garg; D. Marculescu. “HaDeS: architectural synthesis for heterogeneous dark silicon chip multi-processors”. In: DAC, 2013, 7p.

[11] G. Castilhos; M. Mandelli; G. Madalozzo; F. Moraes “Distributed Resource Management in NoC-Based MPSoCs with Dynamic Cluster Sizes”. In: ISVLSI, 2013, pp. 153-158

[12] A. Martins; D. Silva, G. Castilhos, T. Monteiro, F. Moraes. “A method for NoC-based MPSoC energy consumption estimation”. In: ICECS, 2014, pp. 427–430.