Power Consumption Reduction in MPSoCs through DFS

Thiago Raupp da Rosa, Vivian Larréa, Ney Calazans, Fernando Gehm Moraes
PUCRS – FACIN – Av. Ipiranga 6681 – Porto Alegre – 90619-900 – Brazil
thiago.raupp@acad.pucrs.br, vivian.larrea@acad.pucrs.br, ney.calazans@pucrs.br, fernando.moraes@pucrs.br

Abstract—**The use of power management techniques is mandatory in embedded devices, which must provide high performance with low energy consumption. Due to the high variability present in the applications workload executed by these devices, this management should be executed dynamically. The use of traditional dynamic voltage and frequency scaling (DVFS) techniques proved to be useful in several scenarios to save energy. Nonetheless, due to technology scaling that limits the voltage variation and slow response of the DVFS schemes, the use of such technique may become inadequate. As alternative, the use of dynamic frequency scaling (DFS) may provide a good trade-off between power savings and power overhead. This paper proposes a distributed DFS scheme for NoC-based MPSoCs. Both NoC and PEs have an individual controlling scheme. The DFS scheme for PEs takes into account its computation and communication load to dynamically change the operating frequency. In the NoC, the DFS controller uses packet information to decide the router operating frequency. Real and synthetic applications were used to evaluate the proposed scheme. Results show that the number of executed instructions is reduced up to 41%, with an execution time overhead up to 18%. The power dissipation is reduced in PEs up to 26% and in the NoC up to 76%.** *(Abstract)*

Keywords-DFS, MPSoC, NoC, power management (key words)

I. Introduction and Related Work

Controlling the supply voltage and the operating frequency can reduce the energy consumption in CMOS circuits. The operating frequency has a linear impact on energy consumption. On the other hand, voltage has a quadratic impact, being the most used factor to reduce energy consumption. Controlling these two variables at runtime is the basis of DVFS techniques. However, the technology scaling, coupled with the increasing manufacturing process variability, may interfere in the design of DVFS techniques. These variations can result in nominally correct DVFS schemes failing to meet frequency or power targets [1][2]. Therefore, DVFS techniques must cope with design variability in nanoscale technologies to guarantee correct system behavior.

Related work research showed that the DVFS technique might be applied in several architectures, from single CPUs to NoC-based MPSoCs. Architectures targeting single CPUs present DVFS monitoring parameters in hardware (CPU temperature [3], power supply current [4]), and software (application history [5]), being the CPU workload one of the most used parameter. On the other hand, most works addressing NoC architectures use the load in the buffers as monitoring parameter [6][7]. In multi-processed architectures, the monitoring parameters are mostly implemented in software (tasks deadlines [8], application profile [9]), being [10] a work that uses the load in the communication queues as monitoring parameter.

Table 1 summarizes the characteristics of works that present solutions for DVFS in NoC-Based MPSoCs. All works present solutions for DVFS only in the processing element. Apart from Beigné et. al. [14], the controlling algorithm is related to the task characteristics. Also, most of the works do not implement the hardware support needed for DVFS. Puschini [12] uses a platform that already supports DVFS. Goossens [11] implements a frequency scaling mechanism in FPGA, but do not use voltage scaling, while [13] use a simulation platform to generate the results.

Table 1 – DVFS techniques comparison for NoC-Based MPSoCs.

Author	Monitoring Parameter	Implemen-tation	Hardware Support	Memory Architecture
Goossens [11]	Task Slack	Hardware	Partially Implemented	Shared
Puschini [12]	Task Synchronization, Latency, Temperature	Software	Native	Distributed
Gligor [13]	Task Slack	Software	Not Implemented	Shared
Beigné [14]	-	Hardware	Implemented	Distributed
Present work	Communication load and CPU utilization	Software/ Hardware	Implemented	Distributed

This paper proposes and evaluates a DFS technique with fixed system voltage. Contrary to the presented works, the DFS is applied to the PEs (processing elements) and to the NoC. In the PE, the frequency is adjusted according to each processor workload and communication load. In the NoC, for each packet received by the router, the proposed scheme is able to choose the router operation frequency.

This paper is organized as follows. Section II presents the reference MPSoC platform, modified to cope with the proposed DFS scheme. Section III and IV present the DFS scheme at the processing element and NoC, respectively. Section V presents experiments and obtained results. Finally, section VI concludes the paper and draw directions for future works.

II. System Architecture

The reference MPSoC [15] is a homogeneous NoC-based MPSoC. Each PE contains the following modules: (*i*) a 32-bit Plasma processor (MIPS-like architecture); (*ii*) a local memory (RAM); (*iii*) a DMA module, responsible for transferring the task object code to the memory and messages to/from the NoC from/to the local memory; (*iv*) a network interface (NI). The Hermes NoC is used to interconnect the PEs.

Figure 1 presents the proposed architecture for the PE with DFS controller. Initially, the router-PE interface was modified to work according to the GALS paradigm. This is achieved by adapting the existing local buffer in the NoC and NI to work as a bisynchronous FIFO [16], and introducing two-flop synchronizers in control signals. Besides this hardware modification, the *microkernel* (operating system of the processor) was modified to monitors the CPU utilization and communication queue occupancy, storing them in memory-mapped registers. Based on this information, the DFS controller can take decisions and switch the frequency of the processor dynamically.

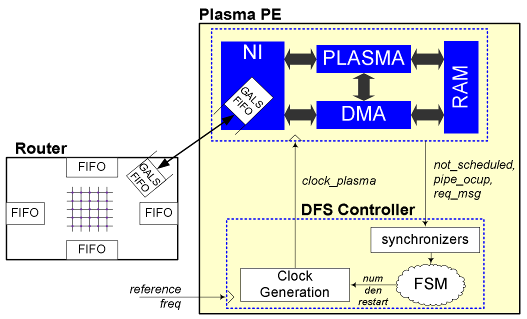

Figure 1. Router-PE GALS interface and the DFS controller responsible for generating the PE frequencies.

It is important to highlight that each individual clock generation module receives the system clock as reference, generating a new clock from it. The benefit of such approach is that the global clock has to feed only the clock generation modules, reducing significantly the global clock load, and hence simplifying the clock tree generation and its power consumption, which is responsible for, at least, 40% of the power consumption in MPSoCs. The global system clock is defined in the following sections as *reference frequency*.

A. Clock Generation

The clock generation module is instantiated at each PE and router. This module uses as input the reference frequency. The principle of the clock generation process is to achieve clock division by simply omitting selected cycles of the reference clock, as Figure 2 illustrates. Initially, inputs *num_i* and *den_i* are natural numbers 3 and 4, respectively. This corresponds to set the frequency of the clock generator to three-fourths (75%) of the reference frequency. In other words, for each *den_i* reference clock cycles, *num*_i cycles are propagated to the output clock. When the frequency needs to be changed, the clock is stopped by asserting the *restart* signal. Finally, releasing the *restart* signal, the new frequency is available at the output clock.

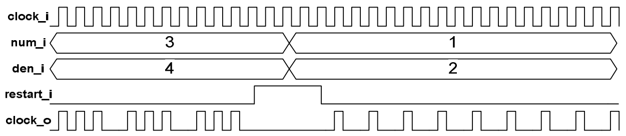

Figure 2. Example of the proposed clock generation process. Signal clock_i is the reference frequency and clock_o is the output of the clock generator.

The main advantages of this clock generation module are the low area overhead and a large set of generated frequencies. For example, for *num_i* and *den_i* being 4-bit values the module takes 107 cells for a 65 nm standard cell library from ST Microelectronics. In this same example, 71 different frequency values can be obtained. In addition, the clock output is always stable, differently to what happens in standard DFS methods, where the time required to stabilize a new frequency can be significant. The proposed module is also glitch free by construction.

III. PE DFS CONTROLLER

The PE DFS controller computes the communication load and CPU utilization level according to values provided by the microkernel. The *key parameter* to control the PE frequency is the communication load. Each PE contains a global vector in the local memory, named *pipe*, controlled by the *microkernel*, which stores the messages to be sent for the remote PEs. As applications are modeled as task graphs, monitoring the *pipe* occupation enables to adjust the data flows by regulating the PE frequencies.

Table 2 summarizes the behavior of the PE DFS controller (Figure 1). The controller evaluates the following parameters:

- *Pending message requests* from other tasks. This situation takes place when the processor is not producing data to its consumer task. When the *microkernel* receives a request to deliver a message to a remote PE, and the message is not in the *pipe*, the signal *req_msg* is asserted.
- *Occupancy of the communication pipe*. If the communication pipe has a high occupancy, the processor is producing messages at a higher rate than the consumer tasks can consume, while the inverse scenario means a lack of produced messages. Upper and lower parameterizable thresholds define the high and low occupancy states, respectively. Occupancy between these values defines an operational state. This parameter, jointly with *pending message requests*, defines the communication load of the PE. The pipe occupation is sampled using a parameterizable interval (in the present implementation it corresponds to 4 time slices).
- *CPU utilization*. When the utilization is low, the CPU is not executing any task or tasks are blocked, e.g., waiting message(s) from other tasks. When the utilization is high, tasks are using the processor at the maximum rate. Two parameterizable thresholds define high, low and operational CPU utilization states.

Table 2. DFS Controller behavior ('↓/↑' mean decrease/increase one frequency step, '↑↑' means increase two frequency steps, '=' means keep frequency unchanged and '-' denotes don't care conditions).

Action in frequency	*Pending Message*	*Current Pipe Occupancy*	*Previous Pipe Occupancy*	*CPU Utilization*
1 - ↓	0	high	-	-
2 - ↓	0	operational	low	-
3 - ↓	0	low	-	low
4 - =	0	operational	operational	-
5 - =	0	low	-	operational
6 - =	1	-	-	low
7 - ↑↑	1	-	-	operat./high
8 - ↑	0	low	-	high
9 - ↑	0	operational	high	-

Frequency decreases in three situations: (i) the *pipe* is almost full (action 1 of Table 2); (ii) the *pipe* occupation is increasing, i.e. in the previous evaluation its state was low and the present state is operational (action 2); (iii) the *pipe* occupation is almost empty and the CPU usage is low, meaning that even at a lower frequency the data in the *pipe* is being consumed (action 3).

Frequency increases in three situations: (i) existence of pending messages with operational or high CPU utilization (action 7) – the clock generator increases the frequency in two steps; (ii) the *pipe* is almost empty and the CPU has high utilization (action 8); (iii) the *pipe* occupation is dropping, e.g. in the previous evaluation its state was high and the present state it is operational (action 9).

Also, the DFS controller implements a communication mechanism to balance power consumption and performance. It takes into account that a consumer processor must receive the data in a frequency that is not inferior to its operating frequency. Thus, if the producer processor is operating at higher frequency than the consumer processor, the message can be sent at a lower frequency to save power. On the other hand, if the producer is operating at a lower frequency than the consumer, its frequency should be temporally increased to not penalize the consumer performance. A set of signals {*msg_transfer*, *dma_active*, *rem_freq*} are responsible for accomplishing this action.

The reception of a packet with REQUEST_MESSAGE service (data request from other task mapped in another processor) asserts the signal *msg_transfer*. This packet contains, among other fields, the frequency of the PE requesting data. Such frequency is coded in the signal *rem_freq* (remote frequency). According to the *rem_freq* the following situations may occur:

- If *rem_freq* is lower than the PE frequency, the PE frequency does not need to be changed.
- If *rem_freq* is higher than the PE frequency and there is data in the *pipe*, the PE frequency is set to the *rem_freq* during the transmission of the message. The PE frequency returns to the original frequency monitoring the *dma_active* signal. When this signal returns to zero, it means the DELIVER_MESSAGE transmission has finished.
- If *rem_freq* is higher than the PE frequency and there is no data in the *pipe*, the rules of the FSM are used. In this case, it is highly probable that the FSM applies the rule 7, increasing by two steps the processor frequency, producing data faster to the consumer.

To conclude the PE DFS controller, all control packets are injected into the NoC at the reference frequency, making such packets available to the target processor as soon as possible.

A. Multitask Support

In the work of Goossens et al. [11] the proposed DFS scheme saves the task context when it is preempted. Later, when the task is rescheduled, the frequency in which it was operating before being preempted is set. Thus, for each task executing in the processor, the controller may change the operating frequency. However, this approach may present problems when several tasks with different characteristics are being executed in the same processor. In this case, at each time slice, the processor may have its frequency drastically changed, e.g. from the highest to the lowest available frequency. Thus, the controlling scheme must adjust the processor to operate in a frequency that satisfies the requirements of all tasks being executed, changing the frequency as less as possible.

To avoid such issue, a simple solution in software was adopted: limitation of the amount of resources that a given task can use. Regarding to the CPU utilization, no modifications are needed, since the *round robin* scheduler provides the same amount of time for each task to execute, and the tasks are preempted in case of blockages. On the other hand, the *pipe* utilization needs to be controlled. To avoid the use of entire *pipe* by a single task, the *microkernel* scheduler was modified to also take into account the *pipe* occupation of the task to be scheduled. Thus, the task is scheduled only if it is not using more than a given percentage of the *pipe*.

Nevertheless, this exception is used only when there is more than one task running in the processor, i.e. in mono-task execution the task can use the entire *pipe* if needed. With this modification the DFS controller is able to adjust the PE frequency also in multi task scenarios.

IV. NoC DFS Controller

Designing a router that may work at multiple frequencies and communicate with neighbor routers working at different frequencies requires adaptations on all input/output buffers. All buffers (*local*, *east, west, north* and *south*) of the router have been replaced by bisynchronous FIFOs to synchronize the communication. In addition, it was necessary to modify the router arbiter, which is responsible for requesting the frequency switching.

Two strategies were evaluated for the router architecture with DFS: (*i*) decentralized and (*ii*) centralized. In the decentralized approach it is necessary to control each input buffer independently. This approach is more flexible, enabling different packets to be sent at different frequencies. However, it requires one frequency controller and one clock generation module per buffer, increasing area and power overheads. On the other hand, the centralized approach induces lower power and area overhead, since only one DFS controller is necessary, but decreases the flexibility of the DFS approach. Experiments conducted in the MPSoC favored the centralized approach, since the packet injection ratio by PEs is frequently small (below 10%). Therefore, the present work adopted the centralized approach to control each router frequency.

The router architecture with a centralized DFS controller is presented in Figure 3. The DFS controller is responsible for defining the router frequency. It receives the frequency information from each input buffer and the arbiter control signals (*clk_change*, *activity*, *header*, *selection*, *ack_header*), to set the correct operation frequency of the router (*clk_r*).

Each input buffer obtains its frequency from the received packets, which carries in the header field the frequency value that it needs to be transmitted (Figure 4). This value is an 8-bit array, 4 bits for numerator and 4 bits for denominator, of the required frequency level. This field was designed to cope with the clock generation scheme presented in section II.A.

When the packet is received, the input buffer extracts the frequency information and provides it to the DFS controller. Next, the buffer sends a routing request to the arbiter and starts waiting the acknowledgement from the arbiter. When the arbiter receives the routing request, it sends a frequency switching request to the DFS controller, and starts waiting for the clock synchronization signal. Then, using the information provided by all buffers and the frequency switching request, the controller check from which port the routing request was made, and stops the provided clock.

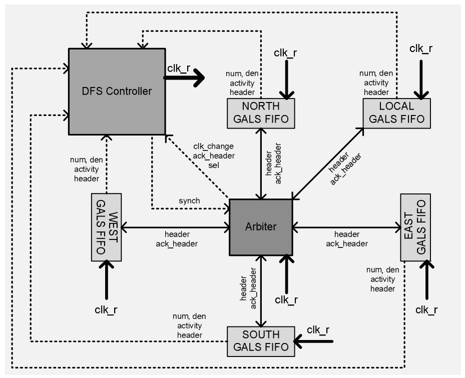

Figure 3. Router architecture with DFS controller.

1st Flit			2nd Flit	
4 bits	4 bits	8 bits	16 bits	
Num	Den	Target	Size	Payload...

Figure 4. Packet with frequency information.

The decision of changing the frequency is made comparing the frequency required by the requesting input buffer with the current router frequency. If the required frequency is lower than the current frequency, the controller only informs the arbiter that the clock is synchronized, without changing the frequency. If the required frequency is higher than the current frequency the controller replaces the values of numerator and denominator of the clock generation module and, after releasing the *restart* signal of clock generation module, informs to the arbiter that the clock is synchronized. Lastly, the arbiter routes the packet.

Nevertheless, when no traffic is being routed, the router slow downs its frequency, going to the minimum available frequency. To implement this mechanism, it is necessary to monitor the router activity. This is achieved by monitoring the buffers occupancies and packet receptions. The inactivity comes from empty buffers and no packet reception in all ports of the router. Also, after finishing the packet transmission, the buffer reset the values of numerator and denominator to the minimum available frequency. This is done to avoid that the router stalls its operating frequency after finishing the transmission of high frequency packets.

Thus, the NoC frequency controller is able to switch the router frequency for each traffic being routed. Also, the controller can identify the router activity and put the router in a low power mode in case of inactivity, saving a significant amount of energy. Yet, when two or more traffic are passing through router, the controller will always select the higher frequency between them, avoiding performance loss in the network.

V. Experimental Results

Several applications were evaluated using the proposed DFS scheme, being presented results related to synthetic (6 tasks pipeline) and real (VOPD and MPEG) benchmarks. The MPSoC was described in VHDL and synthesized using *Cadence* tools targeting a 65 nm standard cell library. Power results were obtained using Encounter RTL Compiler from Cadence. Power estimation is performed using the switching activity obtained through gate level simulation. Nine levels compose the set of available frequencies, from 6.67 to 100 MHz (reference frequency). The other 7 frequencies are 90%, 75%, 60%, 50%, 40%, 25% and 10% of the reference frequency. The parameters *num_i* and *den_i* to achieve these frequencies are defined in a table of the PE DFS controller.

A. Six Tasks Pipeline – one task mapped per PE

The *six tasks pipeline* has 6 tasks, named from T_A to T_F. This is a data flow application, where T_A sends data to T_B, T_B sends data for T_C, and so forth. Two scenarios were evaluated using a 3x3 MPSoC: (1) all tasks having the same computation workload; (2) three different computation workloads: low (T_B and T_E), medium (T_A and T_D) and high (T_C and T_F).

In scenario 1, with same computation workload, all tasks except T_A and T_F had their frequencies increased to the reference frequency. Since T_A and T_F have lower communication load compared to the other tasks, their frequency stabilizes around 60% of the reference frequency.

Figure 5 shows the frequency behavior of each task for scenario 2.

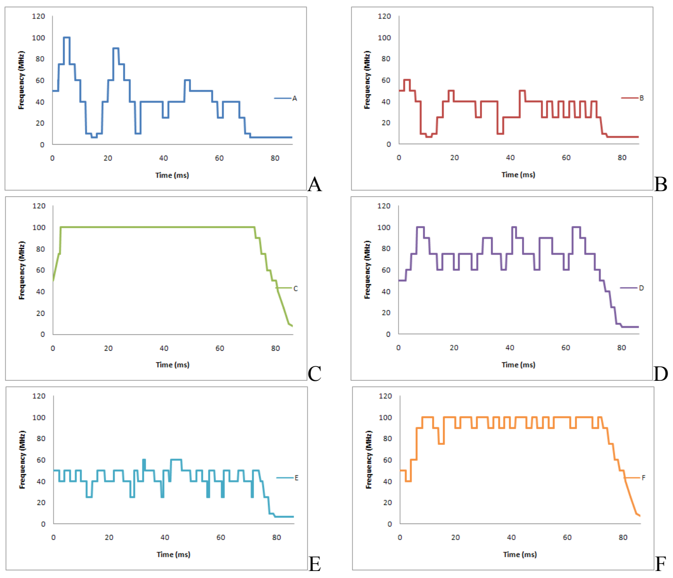

Figure 5. Frequency of PEs running the tasks belonging to the six tasks pipeline application, with different tasks workload.

Tasks with higher computation workload (T_C and T_F) had their frequencies increased to the reference frequency. T_F presented a small variation due to its lower communication load. Tasks with medium computation workload (T_A and T_D) had their frequencies increased in the beginning of the simulation. While T_D kept the frequency around 75% of the reference frequency, T_A presents an oscillation in the beginning of the simulation. This is explained by the time that T_C needs to increase its frequency and start consuming messages from T_B, which also reflects in T_A. Later, T_A had its frequency stabilized around 40% of the reference frequency due to the lower communication load, compared to T_D. Tasks with low computation workload (T_B and T_E) had their frequencies decreased to around 40% of the reference frequency. This is an expected behavior since these tasks need less time to produce data. The execution time overhead, compared to the execution with the whole system executing at the reference frequency is around 12% in the first scenario and 3% in the second scenario, as shown in Table 3 (last column). The number of executed instruction is reduced by 4.4% and 34.5% respectively. The reduction in the number of executed instructions comes from processors that reduced their frequency, resulting in a smaller number of executed instructions by the *microkernel* when there is no task to be scheduled.

Table 3. Six Tasks Pipeline Evaluation.

Scenario		Instructions (in millions)	Reduction (Instruct.)	Execution Time (ms)	Overhead (Exec. Time)
1	Without DFS	20.00	-	41.06	-
	With DFS	19.12	4.40%	45.96	**11.93%**
2	Without DFS	36.00	-	70.03	-
	With DFS	23.56	34.56%	72.29	**3.23%**

Table 4 shows the obtained results in terms of power dissipation for the second scenario. The total power reduction in the system was around 25% (5.7% by NoC and 19.3% by PEs). The power reduction in CPUs is similar to the reduction in number of executed instructions (~32% and ~34%), however, as the RAM memory had lower power reduction (~16%) and it is the module that has the highest power dissipation, the PEs presented a power reduction of 20.85%. Although the achieved NoC power reduction is around 73%, its contribution in the total power reduction is only around 5%.

Table 4 – Power dissipation results for six tasks pipeline application.

	Total Power (mW)				
	RAM	CPU	DFS Controller	PEs	NoC
With DFS	83.04	17.97	2.46	105.73	3.15
Without DFS	99.08	26.64	-	133.58	11.48
Reduction	**16.19%**	**32.53%**	**-**	**20.85%**	**72.58%**

Figure 6 shows the relation for the power consumption with and without the proposed DFS scheme. It can be seen that the contribution of the PEs power dissipation is much larger than the NoC. Considering only the obtained power reduction, the contribution of PEs power reduction is more than 90% and the contribution of the NoC is less than 10 %.

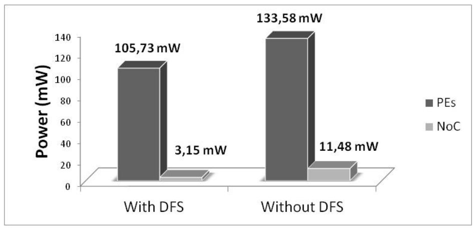

Figure 6. PEs and NoC power dissipation comparison for six tasks pipeline application.

B. VOPD – one task mapped per PE

This application was evaluated in an MPSoC of size 4x4, using 12 processors. The task mapping was performed so that the initial tasks are the first to be allocated (*VLD*, *ARM*), and the final tasks (*PAD*, *VOP-Rec* and *VOP-Mem*) are the last one to be mapped. Table 5 presents the results in terms of number of executed instructions and execution time. The proposed scheme reduced the number of executed instructions by 27.70% and induces 18.33% overhead in execution time.

Table 5 – Number of executed instructions and execution time results.

Scenario	Execution Time (ms)	Executed Instructions (in millions)
No DFS	266.30	313.90
With DFS	315.10	226.94
Difference	**+18,3%**	**-27.7 %**

Similarly to the first benchmark, the reduction in power dissipation is nearly the same of the number of executed instructions, especially when only the CPUs are taken into account (27.7% and 26.6% respectively). The power reduction in the NoC was around 75%. The relation between NoC dissipation and PEs dissipation is illustrated in Figure 7. Again, the reduction of NoC power dissipation reflects in a small fraction of the total power reduction. In this application, the total power reduction for the whole system is 29.53%, being 5.54% the NoC contribution and 23.99% the PEs contribution.

Table 6 – Power dissipation results for VOPD application.

	Total Power (mW)				
	RAM	CPU	DFS Controller	PEs	NoC
With DFS	163.52	34.55	4.92	206.98	5.48
Without DFS	219.93	45.82	-	279.31	22.18
Reduction	**25.31%**	**26.60%**	-	**25.89%**	**75.29%**

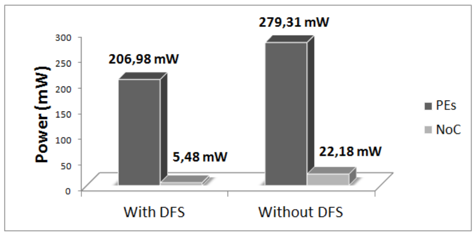

Figure 7 – PEs and NoC power dissipation comparison for VOPD application.

C. Partial MPEG Filter – multitask execution

This application was evaluated in three different scenarios. The first scenario evaluates tasks with similar characteristics mapped in the same processor, while the second scenario evaluates tasks with different characteristics mapped in the same processor. The third scenario evaluates the MPEG execution in the presence of another application in the system (disturbing application).

The reduction in the number of executed instructions and execution time overhead are presented in Table 7. The first two scenarios presented similar reduction in the number of executed instructions, and the third scenario presents a smaller reduction due to the concurrent application executing with the disturbing application. The reduction in the number of executed instructions ranges from 25% to 41%, while the execution time overhead ranges from 8.9% to 12%.

Table 7 – Number of executed instructions and execution time results for the MPEG application in multi task execution.

Scenario		Instructions (in millions)	Reduction (Instruct.)	Execution Time (ms)	Overhead (Exec. Time)
1	Without DFS	23.03	-	85.45	-
	With DFS	13.56	41.12%	95.63	**11.91%**
2	Without DFS	22.97	-	85.40	-
	With DFS	13.55	40.93%	93.01	**8.91%**
3	Without DFS	37.99	-	85.54	-
	With DFS	28.14	25.93%	95.80	**11.99%**

Table 8 present results for power dissipation (3rd scenario). The power reduction in PEs was around 10%, while in the NoC it was around 65%. Similarly to the previous results, considering only the savings in the CPUs, the reduction in the number of executed instructions is similar to the power savings. However, the small saving in the RAM memory reduces the average reduction in PEs.

Table 8 – Power dissipation results for the third scenario of MPEG application in multi task execution, with and without the proposed DFS scheme.

	Total Power (mW)				
	RAM	CPU	DFS Controller	PEs	NoC
With DFS	75.64	16.90	2.10	96.77	2.42
Without DFS	78.50	22.87	-	107.62	7.02
Reduction	3.64%	26.12%	-	10.08%	65.56%

The relation between NoC dissipation and PEs dissipation is illustrated in Figure 8. As presented before, the reduction in NoC power dissipation contributes only as a small percentage of the total power savings. The total power savings for the whole system was 13.48%, being 9.47% from PEs and 4.01% from the NoC.

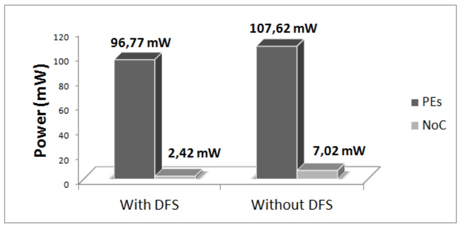

Figure 8 – PEs and NoC power dissipation comparison for MPEG and 4 tasks pipeline applications in multitask execution (scenario 3).

VI. CONCLUSION

This work proposed a DFS scheme for NoC-Based MPSoCs and evaluated it through synthetic and real application scenarios. The DFS scheme is applied in the PE, taking into account the communication and utilization loads, and in the NoC, using packet information to adjust the routers frequency independently. Also, a clock generation scheme was presented and used to enable the DFS at routers and PEs.

The results for mono task execution show that the proposed DFS scheme is able to tune the frequency of each processor in the MPSoC according to the application characteristics. Decrease the operating frequency of a given processor, leads to decrease the power dissipation. However, with the induced execution time overhead, the application performance is penalized. Thus, the DFS controller scheme must induces as less as possible execution time overhead, to maximize both power and energy savings. Considering all conducted experiments (some not presented due to paper length limitation), the reduction in the number of executed instructions ranges from 4.4% to 65.1%, while the execution time overhead ranges from 3.2% to 18%. It is important to highlight that the NoC power savings was around 70% for all applications. This surprisingly result is easily understandable, since most of the time PEs are computing data and not transmitting it. In this way, routers are operating at a lower frequency most of time. So, simple DFS schemes may be applied to NoCs to achieve important power savings.

The reductions observed for multi task execution are smaller to the mono task execution, due to the resource sharing in PEs and higher NoC traffic. The reduction in the number of executed instructions ranges from 25% to 41%, and the execution time overhead ranges from 0.63% to 11.99%. The average NoC power dissipation was 65.5%, against the 70% in mono task execution. Still, the proposed DFS scheme for the NoC presents significant power savings.

Future works include: (*i*) evaluate variable sampling period for the DFS controller; (*ii*) implement higher level methods to evaluate power in PEs and NoC; (*iii*) improvement of the proposed multi task policy; (*iv*) link the proposed method to application deadlines, enabling the DFS controller to select the appropriate PE frequency; (*v*) evaluate the method with more complex processors.

ACKNOWLEDGMENTS

The Author Fernando Moraes acknowledge the support of CNPq and FAPERGS, projects 301599/2009-2 and 10/0814-9, respectively.

REFERENCES

[1] Herbert, S.; Marculescu, D. "Variation-aware dynamic voltage/ frequency scaling". In: HPCA, pp. 301-312, 2009.

[2] Garg, S.; Marculescu, D.; Marculescu, R.; Ogras, U."Technology-driven limits on DVFS controllability of multiple voltage-frequency island designs: A system-level perspective". In: DAC, pp. 818-821, 2009.

[3] Shu, L.; Li, X. "Temperature-aware energy minimization technique through dynamic voltage frequency scaling for embedded systems". In: ICETC, pp. V2-515-V2-519, 2010.

[4] Pourshaghaghi, H.R.; de Gyvez, J.P. "Dynamic voltage scaling based on supply current tracking using fuzzy Logic controller". In: ICECS, pp. 779-782, 2009.

[5] Salehi, M. E.; Samadi, M.; Najibi, M.; Afzali-Kusha, A.; Pedram, M.; Fakhraie, S. M. "Dynamic Voltage and Frequency Scheduling for Embedded Processors Considering Power/Performance Tradeoffs". IEEE Transactions on Very Large Scale Integration Systems, vol.19, no.10, pp. 1931-1935, Oct. 2011.

[6] Garg, S.; Marculescu, D.; Marculescu, R. "Custom Feedback control: Enabling truly scalable on-chip power management for MPSoCs," In: ISLPED, pp. 425-430, 2010.

[7] Yin, A. W.; Guang, L.; Nigussie, E.; Liljeberg, P.; Isoaho, J.; Tenhunen, H. "Architectural Exploration of Per-Core DVFS for Energy-Constrained On-Chip Networks". In: DSD, pp.141-146, 2009.

[8] Chabloz, J. M.; Hemani, A. "Distributed DVFS using rationally-related frequencies and discrete voltage levels". In: ISLPED, pp. 247-252, 2010.

[9] Kong, J.; Choi, J.; Choi, L.; Chung, S. "Low-Cost Application-Aware DVFS for Multi-core Architecture". In: ICCIT, pp. 106-111, 2008.

[10] Alimonda, A.; Carta, S.; Acquaviva, A.; Pisano, A.; Benini, L. "A Feedback-Based Approach to DVFS in Data-Flow Applcations". IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 28, no. 11, pp. 1691-1704, Nov. 2009.

[11] Goossens, K.; She, D.; Milutinovic, A.; Molnos, A.; "Composable Dynamic Voltage and Frequency Scaling and Power Management for Dataflow Applications." In: DSD, pp. 107-114, 2010.

[12] Puschini, D.; Clermidy, F.; Benoit, P.; Sassatelli, G.; Torres, L. "Temperature-Aware Distributed Run-Time Optimization on MP-SoC Using Game Theory". In: ISVLSI, pp.375-380, 2008.

[13] Gligor, M.; Fournel, N.; Petrot, F. "Adaptive Dynamic Voltage and Frequency Scaling Algorithm for Symmetric Multiprocessor Architecture". In: DSD '09, pp. 613-616, 2009.

[14] Beigné, E.; Clermidy, S.; Miermont, P. “Dynamic Voltage and Frequency Scaling Architecture for Units Integration within a GALS NoC”. In: NOCS, 2008.

[15] Carara, E.A.; de Oliveira, R.P.; Calazans, N.L.V.; Moraes, F.G. "HeMPS - a framework for NoC-based MPSoC generation". In: ISCAS, pp. 1345-1348, 2009.

[16] Chelcea, T.; Nowick, S. "A low latency FIFO for mixed-clock systems". In: Computer Society Workshop on VLSI, pp. 119-126, 2000.