Energy Consumption Reduction with Low Computational Needs in Multicore Systems with Energy-Performance Tradeoff

Sylvain Durand, Nicolas Marchand

▶ To cite this version:

HAL Id: hal-00404053

https://hal.archives-ouvertes.fr/hal-00404053

Submitted on 6 Apr 2011

Energy Consumption Reduction with Low Computational Needs in Multicore Systems with Energy-Performance Tradeoff

Sylvain Durand and Nicolas Marchand

Abstract— A two voltage level electronic device is interesting because the clock frequency and the supply voltage level could be reduced (respecting certain rules) in order to decrease the energy consumption. We proposed in a previous paper a robust control architecture to deal with this power-performance tradeoff and we are now interested in extending this principle for several devices which works together since they are all supplied with the same voltage and clock frequency. Thus, an intuitive multicore control strategy which duplicates the whole monocore architecture as much as devices is compared with a second strategy where the duplication is reduced as much as possible. It appears that the proposal clearly gives a low control computational needs with the same reduction of the energy consumption.

I. INTRODUCTION

An energy-performance tradeoff is required in many embedded electronic systems. Actually, three power consumption sources exist in CMOS circuits [4], which could be sorted into a dynamic consumption from switching of electrical gates and a static consumption from short circuit and leakage currents:

$$\begin{aligned} P &= P_{switching} + P_{short\ circuit} + P_{leakage} \\ P &= K_{dyn}f_{clk}V_{dd}^2 + K_{sc}f_{clk}V_{dd} + K_{leak}V_{dd} \end{aligned} \quad (1)$$

It appears that the consumption could be reduced by decreasing V_{dd}, i.e. the supply voltage, or f_{clk}, i.e. the clock frequency. However, decreasing only the frequency will decrease the power consumption and results in a slower running task but the total energy consumption will remain unchanged [12]. The voltage has hence to be reduce in order to decrease the energy consumption. Furthermore, the supply voltage is the dominant term especially because the dynamic power is the most important part in (1). In other words, decreasing the voltage will almost quadratically decrease the energy consumption. Unfortunately, this drop will decrease the computational speed (because of the propagation delay of transistors) and controlling the supply voltage is hence a power-delay tradeoff: the power consumption decreases while the delay increases. That is why the supply voltage and the clock frequency have to be controlled together to guarantee the critical path (the longest electrical path a signal can travel to go from a point to another of the circuit). Clearly, it is required to decrease the clock frequency before decreasing the supply voltage and, respectively, increase the supply voltage before increasing the clock frequency.

S. Durand is with NeCS Project-Team, INRIA - GIPSA-lab - CNRS, Grenoble, France, sylvain.durand@inrialpes.fr

N. Marchand is with NeCS Project-Team, INRIA - GIPSA-lab - CNRS, Grenoble, France, nicolas.marchand@gipsa-lab.inpg.fr

A good energy-performance tradeoff could be achieved using a commonly used approach in embedded systems: the Dynamic Voltage and Frequency Scaling (*DVFS*). This method consists in adapting the voltage and the frequency to the computational load and leads up to an important energy consumption reduction (regarding the application) [10]. Furthermore, it seems that most of the applications could run with a reduced voltage [2], [3]. Thus, several behaviors are known to minimize the energy consumption. Firstly, each task has to be considered independently and its execution time has to fit with the deadline. Moreover, selecting some suitable voltage levels leads to a drastic energy reduction even if the number of levels is very small [7]. The supply voltage has to be reduced as much as possible and the frequency clock adapted to the computational load to minimize the energy consumption [11].

Based on these different rules, we proposed in [5] a robust strategy to control the clock frequency and the supply voltage level of an electronic device. The proposal leads to minimize the energy consumption while guaranteeing a good computational performance. We are now interested in extending this principle to several devices which works together (with the same voltage and frequency domain) but where each device has to deal with a different load. In the following section, we first propose to bring back the monocore system architecture and summarize its control strategy. In section III, the multicore architecture is then presented and two control strategies are detailed: a first intuitive one which duplicates the monocore principle as much as devices and a second strategy which reduces considerably the computational needs. Finally the two controllers are compared in section IV in term of energy consumption and control computational needs.

II. MONOCORE SYSTEM PRINCIPLE

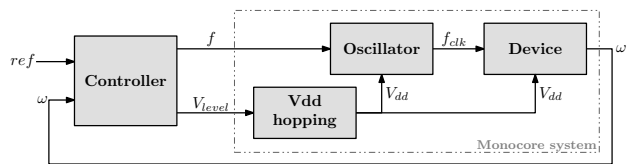The system architecture with only one device to control is shown on Figure 1.

Fig. 1. Monocore system architecture

The ***Device*** is the system to control. It usually runs at nominal supply voltage and constant clock frequency but

these quantities will now dynamically vary in order to reduce the energy consumption. That is possible introducing a closed-loop with a controller to monitor the activity of the device (its computational speed ω) and to adapt the supply voltage and the clock frequency regarding the computational load ref provided by the operating system for each task.

The ***Oscillator*** and the ***Vdd-hopping*** are the two actuators used in some DVFS systems. They respectively provide the clock frequency and the supply voltage to the device:
- The oscillator could be a ring oscillator [6].
- The Vdd-hopping principle is described in [1]. Two voltage levels are available (V_{low} and V_{high}) and the one or the other could be achieved (with a certain transition time and dynamics that depends upon the internal controller of the Vdd-hopping) regarding the V_{level} input signal: $V_{level} = level_{low}$ to require the low voltage and respectively $level_{high}$ for the high voltage.

The ***Controller*** has to provide the control signals to the actuators. Actually, the controller can be divided into two parts, as depicted on Figure 2:
- The ***computational speed controller (CSC)*** provides the computational speed set point ω_{sp}. Thus, from some task informations - for each task T_i the operating system provides the computational load (i.e. the number of instructions C_i) and the time before the task has to be executed (i.e. the deadline N_i) - a fast predictive control law permits to calculate the best speed set point in order to minimize the penalizing high voltage running time (and so the energy consumption) while guaranteeing the computational performance.
- Then the ***frequency and voltage level controller (FVC)*** fits the measured speed ω with the desired one ω_{sp}, by adapting the frequency f and the voltage level V_{level}.
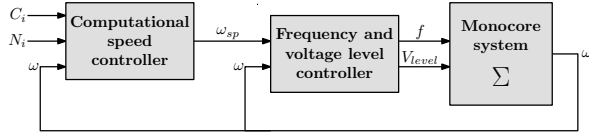

Fig. 2. Monocore controller architecture: a ***computational speed controller (CSC)*** plus a ***frequency and voltage level controller (FVC)***

The whole monocore controller (CSC + FVC) leads to a robust control (see [5] for further details): for a given test bench, the device runs at the penalizing high supply voltage only during 30% of time and an energy consumption reduction of about 20% is achieved. We propose next to adapt this principle to a system with several devices.

III. MULTICORE SYSTEM PRINCIPLE

The system architecture with several devices to control is shown on Figure 3. In fact this system is not so different from the monocore one (presented in section II and shown on Figure 1). Indeed the bases remain the same, with a controller which sends the frequency f and the voltage level V_{level} to the actuators, i.e. a ring oscillator and a Vdd-hopping which respectively provide the clock frequency and the supply voltage to the electronic devices. The main difference is that

there are now N devices to control, which means as many references ref^N given by the operating system (the number of instructions and the deadline for each task) and as many measured computational speeds ω^N as devices. Therefore the controller has to control the whole system but devices do not work independently since they are all supplied with the same voltage V_{dd} and the same clock frequency f_{clk}. The only allowed dimension of freedom is to trigger a device with a ratio of the clock f_{clk} because in fact in practice it is possible to add one or two *NOPs* (i.e. No OPeration) between each instruction in order that the device runs twice or three times slower. For this reason, now the energy controller has to provide the frequency ratios ρ^N anymore.
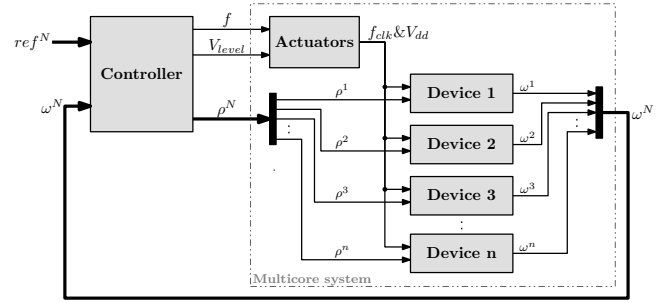

Fig. 3. Multicore system architecture

Notations: ρ^j (lower case indice) denotes the signal ρ of the device j, whereas ρ^N (upper case indice) means that there are N signals ρ, one for each device.

In the two following subsections we will detail two control strategies: a first intuitive one which duplicates the monocore principle as much as devices and a second one which tries to minimize the computational needs of the controller.

A. Multicore control based on full duplication of the monocore control strategy

A first way to control several devices is to duplicate the monocore control strategy (detailed in section II) as much as devices. The resulting multicore architecture is presented on Figure 4 and could be divided in three steps:

1) First, the ***computational speed controller (CSC)*** calculates the speed set points ω_{sp}^N for the whole devices. Thus the set point ω_{sp}^j is independently calculated for each device j, using the task information C_i^j and L_i^j (given by the operating system) and the measured speed ω^j.

2) Then the ***frequency and voltage level controller (FVC)*** independently calculates the frequencies f^N and the voltage levels V_{level}^N usually required to control a single device.

3) Finally a ***frequency ratio controller*** compares the calculated frequencies f^N to deduce the *critical device c*, i.e. the device which needs the maximal frequency to fit with its load. Thus the frequency f and the voltage level V_{level} sent to the actuators are those from the critical device, i.e. f^c and V_{level}^c, and the frequency ratios ρ^N are obtained by doing the ratio between the frequency of the current device f^j and the one of the critical device f^c.
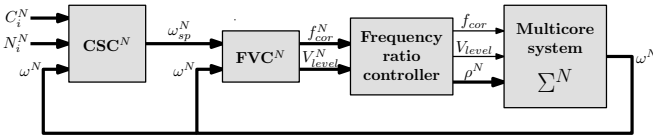
Fig. 4. Multicore control architecture based on full duplication of the monocore control strategy: the **computational speed controller (CSC)** and the **frequency and voltage level controller (FVC)** are duplicated as much as devices and a **frequency ratio controller** calculates the critical frequency and voltage level to deduce the frequency ratios ρ^N

This intuitive strategy guarantees that the tasks are correctly performed for all devices because each device is independently controlled using the monocore strategy. Indeed, the monocore strategy works for one device and we focus the frequency and the voltage level decision on the critical one, i.e. the device which has to treat the task with the highest computational needs. Thus, all the non-critical tasks will be executed with the critical voltage level and a frequency lower or equal to the critical frequency. Moreover, an non-critical device could become the critical one whereas its task requires more and more computational needs.

An improvement could be done for the non-critical devices. Actually, if a device runs at high level then it is forced to the maximal frequency in order to run the shortest possible time at the penalizing high supply voltage (see [5] for further details). A non-critical device - which *a priori* could run at V_{low} - will hence have its frequency forced anyway when the critical device needs to run at V_{high}. For this reason, we propose to force only the frequency of the critical device. However in practice the critical device is not known yet when the frequencies are calculated, i.e. in step 2, because the frequency ratio controller determined it in step 3. Fortunately, a solution consists in using the device which was critical during the previous sampling period, by using the assumption that the critical device does not often change.

This intuitive duplication of the whole monocore principle leads to reduce the energy consumption of several devices working together while guaranteeing their computational performance. Nevertheless, a consequence is that the control computational needs are multiplied as much as devices and the number of variables seriously increases too. That is why we propose next to duplicate only some parts of the monocore control strategy.

B. Multicore control based on partial duplication of the monocore control strategy

This second strategy tries to minimize the control computational needs by not intuitively duplicating all the monocore control strategy. In fact, the frequency ratios ρ^N require to be calculated and so some parts have necessary to be duplicated in order to obtain the N signals. The aim is to repeat the least code as possible. The best solution would be to use the references ref^N (given by the operating system) to deduce the ratios without duplicating any part of the monocore strategy, but these signals are not relevant enough. Indeed, the critical task could not be known only from the number

of instructions and the deadline because the computational load which was already executed is necessary too. Therefore we propose to duplicate the computational speed controller (which seems to have to be repeated anyway). Thus the multicore architecture on Figure 5 is proposed:

1) First, the **computational speed controller (CSC)** provides the speed set points ω_{sp}^N, from which the frequency ratios ρ^N could be obtained since they provide information on the remaining computational load.

2) Then the **frequency ratio controller** compares the whole speed set points ω_{sp}^N to deduce the *critical task c*, i.e. the task which needs the maximal speed to fit with its deadline. Thus the speed set point ω_{sp} and the measured speed ω sent to the FVC are those calculated for the critical task, i.e. ω_{sp}^c and ω^c, and the frequency ratios ρ^N are obtained by doing the ratio between the speed set point of the current device ω_{sp}^j and the one of the critical task ω_{sp}^c.

3) Finally the **frequency and voltage level controller (FVC)** calculates the frequency f and the voltage level V_{level} to send to the actuators only for the critical device, i.e. the device which has to compute the critical task.

With this proposal, only the CSC is repeated and not the FVC anymore. We so hope a reduction of the control computational needs without impacting the gain on the energy consumption.
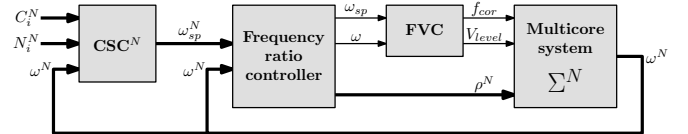

Fig. 5. Multicore control architecture based on partial duplication of the monocore control strategy: only the **computational speed controller (CSC)** is duplicated as much as devices and then a **frequency ratio controller** calculates the critical speed set point which will be used by the **frequency and voltage level controller (FVC)** and deduces the frequency ratios ρ^N

Though all the devices are not independently controlled using the monocore strategy, the computational performances are yet guaranteed for each device. Indeed, with this second architecture the monocore control strategy only guarantees that the critical task will fit with its deadline, since the monocore control strategy is only applied to the critical device. The frequency ratios for the non-critical devices are then calculated from the computational load of the task of each device which is finally adjusted thanks to the CSC. Thus all the non-critical tasks are executed until their deadline anyway, or a task becomes the critical one when its computational needs become the more important one.

C. Discret values of the frequency ratios

One could note that the control algorithms proposed in both previous subsections were developed with *ideal* continuous frequency ratios ρ^N. However, as explained in introduction of the multicore principle, some devices could be triggered with a ratio of the clock frequency f_{clk} by adding NOPs between instructions in order that the device runs slower. This is why the frequency ratios could only be a

discrete value which correspond to the number of NOPs, i.e. $\varrho = \{1; \frac{1}{2}; \frac{1}{3}\}$ for 0, 1 or 2 NOPs respectively added between each instruction (note that the discrete frequency ratios will be called ϱ and the continuous ones ρ).

In order to implement this behavior, we first have to calculate the continuous ratios ρ^N (i.e. $\rho^j = f_{cor}^j / f_{cor}^c$ for the multicore control strategy based on full duplication of the monocore control strategy and $\rho^j = w_{sp}^j / w_{sp}^c$ for the one based on partial duplication). Then, iterations have to be done for each device j in order to deduce the discrete frequency ratio ϱ^j just upper than the value of the continuous ratios ρ^j, as depicted by the below algorithm:

$$
\varrho^j = \begin{cases} 1 & \text{if } \frac{1}{2} < \rho^j \leq 1 \\[4pt] \frac{1}{2} & \text{if } \frac{1}{3} < \rho^j \leq \frac{1}{2} \\[4pt] \frac{1}{3} & \text{if } 0 < \rho^j \leq \frac{1}{3} \\[4pt] 0 & \text{otherwise} \end{cases} \tag{2}
$$

This discrete ratio behavior would lead to a less energy efficient system because the frequencies of the non-critical devices will be higher than required - thanks to (2) - contrary to the continuous case where these frequencies correspond exactly to the desired ones. Moreover, the control computational needs would increase a little bit thanks to the added code required to calculate the discrete ratios ϱ^N.

IV. PERFORMANCE EVALUATION

This section presents some simulation results. The benchmark test is the same for all the simulations, where four devices with a different reference (number of instructions and deadline shown on Figure 6) have to be controlled:

device 1 $\rightarrow$ three tasks to execute: the first task starts with 5 instructions to do in $0.5\mu s$, then a 75 instruction task has to be executed in $2.5\mu s$ and the last one has to compute 10 instructions in $1\mu s$.

device 2 $\rightarrow$ three tasks also: a 15 instruction task to execute in $1.25\mu s$, a task with 50 instructions to do in $2.25\mu s$ and then 5 instructions to execute in $0.5\mu s$.

device 3 $\rightarrow$ a single task of 40 instructions to do in $4\mu s$.

device 4 $\rightarrow$ three tasks again: 10 instructions to compute in $0.75\mu s$, a task with 20 instructions to do in $0.75\mu s$ and a last 40 instruction task to execute in $2.5\mu s$.

First, the simulation results for both control strategies (with *ideal* continuous frequency ratios) are shown on Figures 7 and 8. The top plots show the average speed set point (for guideline), the speed set point ω_{sp}, the measured speed ω and the critical speed ω^c (for guideline) for each device. One could verify that $\omega = \omega^c$ when the device is the critical one (highlighted by the gray areas on plots). Moreover, the supply voltage V_{dd} (which is the same for the whole devices because of the multicore architecture) is shown on the bottom plot. Note that the calculated frequency f or the clock frequency f_{clk} and the voltage level V_{level} are not plotted because they do not provide relevant information: the frequencies are proportional to the speed and the level can be deduced from the voltage.
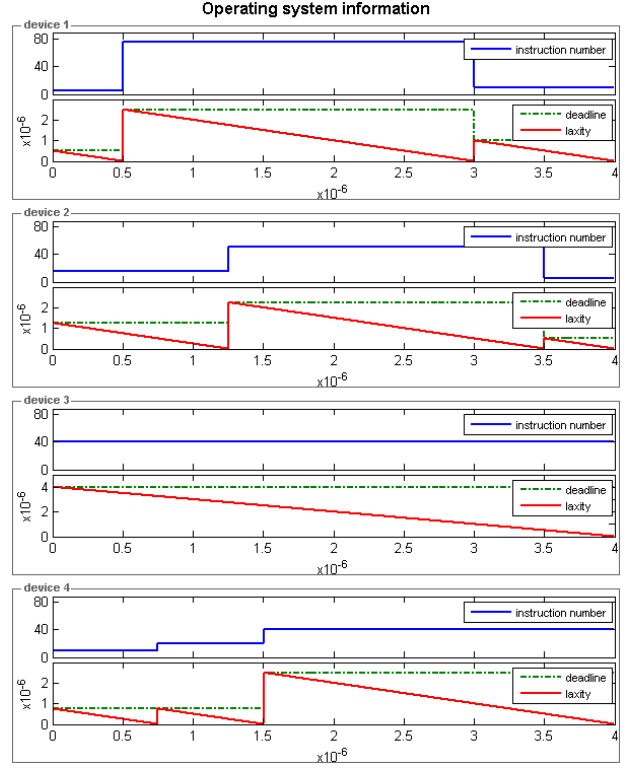

Fig. 6. References used for the simulations: the number of instructions, the deadline and the laxity (the remaining available time to execute the task) for each device

The results are quantified in term of energy consumption and computational needs:

Energy consumption of the system: The energy consumption is calculated in order to have an idea of the reduction achieved thanks to our proposal. Thus, the relation (1) is used and a ratio of this power consumption is added due to the Vdd-hopping principle: 20% more during the voltage transition time and 3% more during the steady state [8]. Finally, an integration during the whole running time gives the total energy consumption.

Computational needs of the controller: The control laws are compared in term of computational needs, i.e. the number of instructions required to calculated the computational speed set points, the frequencies, the voltage levels and the frequency ratios. To do that, we use the Lightspeed Matlab toolbox proposed by T. Minka [9], which provides a number of flops for each instruction.

Moreover, the strategies are compared with a system using the intuitive control strategy (by duplicating the whole monocore control strategy) but without Dynamic Voltage Scaling (DVS): in this case the measured speed tracks the average speed set point and the supply voltage is fixed to the penalizing high voltage, i.e. $V_{level} = level_{high}$.

In both cases, the system runs during more than 50% of the simulation time at low voltage and a reduction of the energy consumption of about 20% is achieved in comparison with a system without DVS. The differences between the two
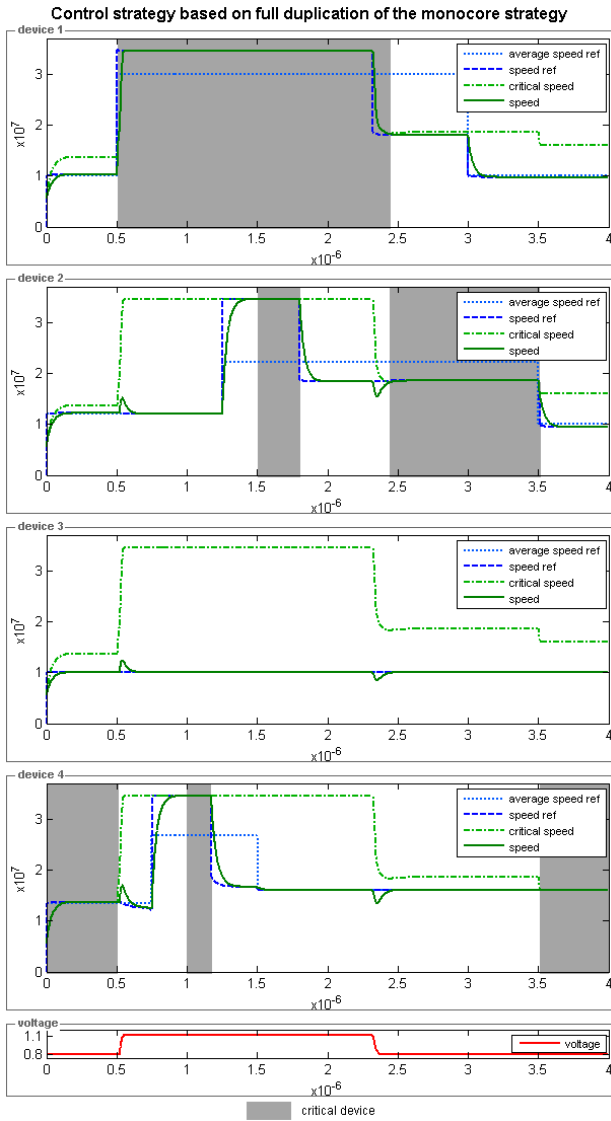
Fig. 7. Simulation results of the multicore controller based on full duplication of the monocore control strategy (with *ideal* continuous frequency ratios): energy consumption of $3.976 \cdot 10^{-5}J$ and computational needs of $5.8 \cdot 10^5 \, flops$, that is 82.2% of energy consumption and 94% of computational needs compared to a controller without DVS
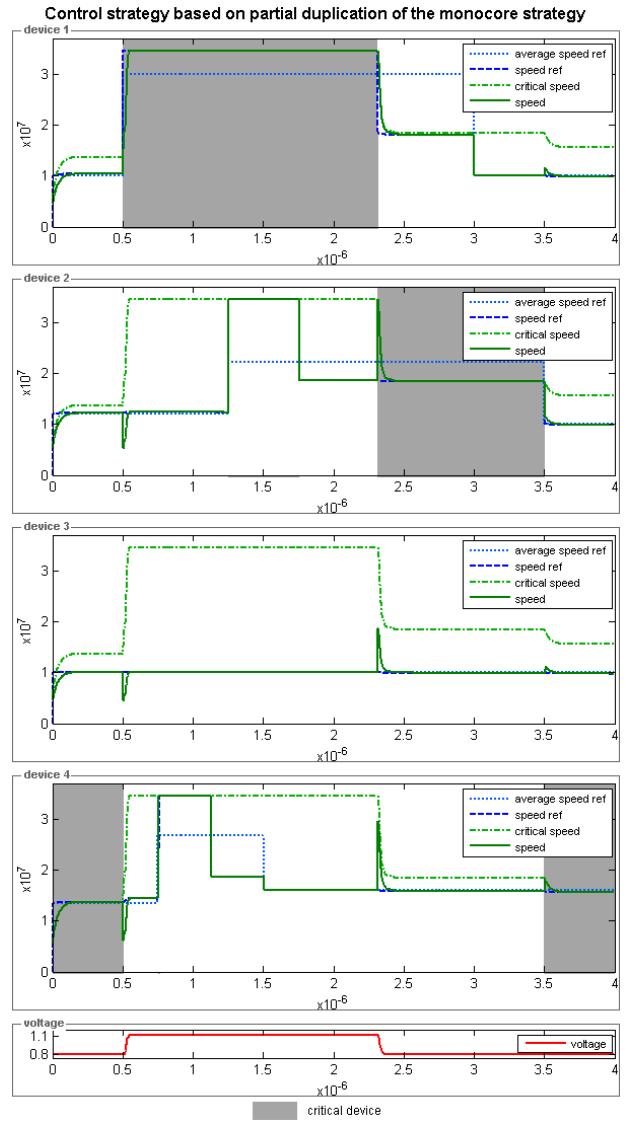
Fig. 8. Simulation results of the multicore controller based on partial duplication of the monocore control strategy (with *ideal* continuous frequency ratios): energy consumption of $3.98 \cdot 10^{-5}J$ and computational needs of $3.8 \cdot 10^5 \, flops$, that is 82.4% of energy consumption and 62% of computational needs compared to a controller without DVS

control strategies are during the voltage transitions and come from the choice of the critical device:

A) For the multicore control strategy based on full duplication of the monocore control strategy, one could see on Figure 7 that the measured speed ω is continuous for all the devices. This is because the ratios ρ^N are obtained from the frequencies f^N independently calculated for each device.

B) For the strategy based on partial duplication, one could see on Figure 8 a discontinuity of the measured speed ω as soon as the critical device changes, such as at time $2.35\mu s$ on device 2. Indeed, the frequency ratios ρ^N are obtained from the speed set points ω_{sp}^N which are switching variables due to their construction (see [5] for further details). Thus the speed set point value of a device could suddenly change and so are the ratios. Nevertheless, the critical frequency -

and so the critical speed - remains continuous.

While the energy consumption is very similar for both strategies, the computational needs is considerably reduced for the second one with a drop of 35% of the number of flops. For this reason, it would be the strategy to use.

Finally we propose to compare the simulation results of the control strategy with low computational cost, on a first hand when the frequency ratios are the *ideal* continuous variables ρ^N and on an other hand when they are the discrete variables ϱ^N described by the algorithm (2). One could immediately remarks than the results, respectively shown on Figures 8 and 9, are quite similar. The main difference is that the measured speed ω does not track the speed set point ω_{sp} in the discrete case as well as in the continuous case. However, the algorithm assures that the speed will be at least upper
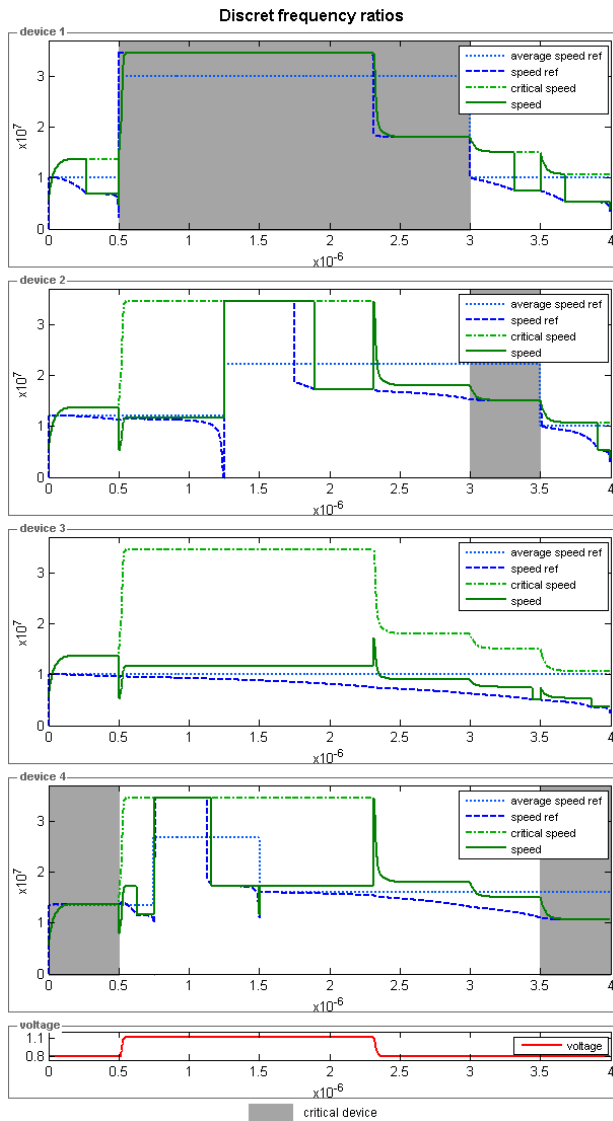
Fig. 9. Simulation results of the multicore controller based on partial duplication of the monocore control strategy (with discrete frequency ratios): energy consumption of $4 \cdot 10^{-5} J$ and computational needs of $4.3 \cdot 10^5 flops$, that is an increase of 1% of energy consumption and 11% of computational needs compared to the controller with *ideal* continuous frequency ratios

than the desired one and so the computational load will be correctly computed. This is why this principle is interesting since it only leads to an increase of less than 1% of the energy consumption and 11% of the control computational needs in comparison with the continuous frequency ratio case (which could not be implemented in practice anyway).

V. CONCLUSIONS AND FUTURE WORKS

This paper proposes architectures to control several devices which work together since they are all supplied with the same voltage V_{dd} and the same clock frequency f_{clk} (or a ratio of this clock). The multicore control strategies are based on the monocore control strategy depicted in [5], where a fast predictive control technique gives a computational speed set point to track in order to minimize the energy consumption

while guaranteeing the computational performance.

While the first multicore control strategy intuitively duplicates the whole monocore architecture as much as devices, the second strategy - the contribution of this paper - tries to minimize as much as possible the duplication in order to decrease the control computational needs. Both architectures lead to a similar gain of energy consumption (compared to a system without DVS mechanism) but an important reduction of the number of flops is achieved with the second one. We finally propose to use discrete frequency ratios which are the only way to implement our controller in practice.

Next steps in this research is to test these control strategies in practice.

VI. ACKNOWLEDGMENTS

REFERENCES

[1] C. Albea, C. Canudas de Wit, and F. Gordillo. Control and stability analysis for the vdd-hopping mechanism. In *Proceedings of the IEEE Conference on Control and Applications*, 2009.

[2] T. Burd and R. Brodersen. Processor design for portable systems. In *The Journal of VLSI Signal Processing*, volume 13, pages 203–221, 1996.

[3] T. Burd, T. Pering, A. Stratakos, and R. Brodersen. A dynamic voltage scaled microprocessor system. In *IEEE International Solid-State Circuits Conference Digest of Technical Papers*, volume 35, pages 1571–1580, 2000.

[4] A. Chandrakasan and R. Brodersen. Minimizing power consumption in digital cmos circuits. In *Proceedings of the IEEE*, volume 83, pages 498–523, 1995.

[5] S. Durand and N. Marchand. Fast predictive control of micro controller's energy-performance tradeoff. In *Proceedings of the 3rd IEEE Multi-conference on Systems and Control - 18th IEEE International Conference on Control Applications*, 2009.

[6] S. Fairbanks and S. Moore. Analog micropipeline rings for high precision timing. In *Proceeding of the International Symposium on Advanced Research in Asynchronous Circuits and Systems*, pages 41–50, 2004.

[7] T. Ishihara and H. Yasuura. Voltage scheduling problem for dynamically variable voltage processors. In *Proceedings of the International Sympsonium on Low Power Electronics and Design*, pages 197–202, 1998.

[8] S. Miermont, P. Vivet, and M. Renaudin. A power supply selector for energy- and area -efficient local dynamic voltage scaling. In *PATMOS'07: 17th International Workshop on Power and Timing Modeling, Optimization and Simulation*, pages 556–565, 2007.

[9] T. Minka. The lightspeed matlab toolbox v2.2. http://research.microsoft.com/˜minka/software/lightspeed/.

[10] T. Pering, T. Burd, and R. Brodersen. Voltage scheduling in the lparm microprocessor system. In *Proceedings of the International Symposium on Low Power Electronics and Design (ISLPED)*, pages 96–101, 2000.

[11] J. Pouwelse, K. Langendoen, and H. Sips. Dynamic voltage scaling on a low-power microprocessor. In *Proceedings of the 7th Annual International Conference on Mobile Computing and Networking*, pages 251–259, 2001.

[12] A. Varma, B. Ganesh, M. Sen, S. Choudhury, L. Srinivasan, and J. Bruce. A control-theoretic approach to dynamic voltage scheduling. In *Proceedings of the International Conference on Compilers, Architecture and Synthesis for Embedded Systems*, pages 255–266, 2003.