Energy Consumption Reduction with Low Computational Needs in Multicore Systems with Energy-Performance Tradeoff
Sylvain Durand, Nicolas Marchand

To cite this version:
Sylvain Durand, Nicolas Marchand. Energy Consumption Reduction with Low Computational Needs in Multicore Systems with Energy-Performance Tradeoff. 48th IEEE Conference on Decision and Control, CDC 2009, Dec 2009, Shanghai, China. pp.5568-5573. hal-00404053

HAL Id: hal-00404053
https://hal.archives-ouvertes.fr/hal-00404053
Submitted on 6 Apr 2011

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L’archive ouverte pluridisciplinaire HAL, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d’enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.
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 monocoarchitecture 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:

\[
P = P_{\text{switching}} + P_{\text{short circuit}} + P_{\text{leakage}}
\]

\[
P = K_{\Omega} f_{\text{clk}} V_{dd}^2 + K_{\text{sc}} f_{\text{clk}} V_{dd} + K_{\text{leak}} V_{dd}
\]

(1)

It appears that the consumption could be reduced by decreasing \( V_{dd} \), i.e. the supply voltage, or \( f_{\text{clk}} \), 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.

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 monocoarchitecture 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 monocoarchitecture 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.

![Monocore system architecture](image-url)

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 \( \omega \)) and to adapt the supply voltage and the clock frequency regarding the computational load \( ref_f \) 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: \( V_{level} = \text{level}_{low} \) to require the low voltage and respectively \( \text{level}_{high} \) 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 \( \omega_{sp} \). 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 \( \omega \) with the desired one \( \omega_{sp} \), 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)](image)

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 \( ref_f^N \) given by the operating system (the number of instructions and the deadline for each task) and as many measured computational speeds \( \omega^N \) 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 \( \rho^N \) anymore.

![Fig. 3. Multicore system architecture](image)

Notations: \( \rho^i \) (lower case indice) denotes the signal \( \rho \) of the device \( j \), whereas \( \rho^N \) (upper case indice) means that there are \( N \) signals \( \rho \) 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 \( \omega^N_{sp} \) for the whole devices. Thus the set point \( \omega^N_{sp} \) is independently calculated for each device \( j \), using the task information \( C^j_i \) and \( L^j_i \) (given by the operating system) and the measured speed \( \omega^j \).

2) Then the frequency and voltage level controller (FVC) independently calculates the frequencies \( f^N \) and the voltage levels \( V^N_{level} \) 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 \( \rho^N \) are obtained by doing the ratio between the frequency of the current device \( f^j \) and the one of the critical device \( f^c \).
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 \textit{a priori} could run at $V_{\text{low}}$ - will hence have its frequency forced anyway when the critical device needs to run at $V_{\text{high}}$. 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.

\textbf{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 $\rho^N$ 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 \textit{computational speed controller} (CSC) provides the speed set points $\omega^N_{sp}$, from which the frequency ratios $\rho^N$ could be obtained since they provide information on the remaining computational load.

2) Then the \textit{frequency ratio controller} compares the whole speed set points $\omega^N_{sp}$ to deduce the \textit{critical task} $c$, i.e. the task which needs the maximal speed to fit with its deadline. Thus the speed set point $\omega_{sp}$ and the measured speed $\omega$ sent to the FVC are those calculated for the critical task, i.e. $\omega^*_c$ and $\omega^c$, and the frequency ratios $\rho^N$ are obtained by doing the ratio between the speed set point of the current device $\omega^j_{sp}$ and the one of the critical task $\omega^c_{sp}$.

3) Finally the \textit{frequency and voltage level controller} (FVC) calculates the frequency $f$ and the voltage level $V_{\text{level}}$ 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.

\textbf{C. Discrete values of the frequency ratios}

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.
the multicore control strategy based on full duplication of
ratios \( \varrho \) for each device \( j \) and the partial duplication. Then, iterations have to
be done for each device \( j \) in order to deduce the discrete frequency ratio \( \varrho^j \) just upper than the value of the continuous ratios \( \rho^j \), as depicted by the below algorithm:

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

(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 \( \varrho^N \).

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 → 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 → 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 → a single task of 40 instructions to do in 4\( \mu s \).

device 4 → 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 Fig-
ures 7 and 8. The top plots show the average speed set point
(for guideline), the speed set point \( \omega_{sp} \), the measured speed \( \omega \) and the critical speed \( \omega^c \) (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.

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

**Energy consumption of the system**: The energy consump-
tion 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 5% 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. \( V_{level} = V_{level high} \).

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
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 $\omega$ is continuous for all the devices. This is because the ratios $\rho^N$ 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 $\omega$ as soon as the critical device changes, such as at time $2.35\mu s$ on device 2. Indeed, the frequency ratios $\rho^N$ are obtained from the speed set points $\omega^{\text{sp}}$ 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 $\rho^N$ and on an other hand when they are the discrete variables $\varrho^N$ 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 $\omega$ does not track the speed set point $\omega^{\text{sp}}$ in the discrete case as well as in the continuous case. However, the algorithm assures that the speed will be at least upper

---

Fig. 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
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

This research has been supported by the NeCS Project-Team (INRIA, GIPSA-lab, CNRS) in the ARAVIS project context. ARAVIS project is a Minalogic project gathering ST Microelectronics with academic partners of different fields, namely TIMA and CEA-LETI for micro-electronics and INRIA for operating system and control. The aim of the project is to overcome the barrier of subscale technologies (45nm and smaller).

REFERENCES


