

Fine-Grained Power Management for Real-Time Embedded Processors

Sascha Uhrig, PhD student
Theo Ungerer, University Professor

University of Augsburg
Dept. of Applied Informatics
Eichleitnerstr. 30
86159 Augsburg, Germany
{uhrig,ungerer}@informatik.uni-augsburg.de

Abstract

This paper proposes a new hardware-based power management technique for multithreaded processors that schedule instructions by the Guaranteed Percentage real-time scheduling scheme, which assigns percentages of the processors utilization to the different threads. Our power management technique uses the sum of the percentages of the currently active threads to control frequency reduction, dynamic voltage scaling, and pipeline gating. Results are obtained by simulating the execution of four threads derived from an autonomous guided vehicle application in a specific processor core – the Komodo processor. Our evaluation showed for that benchmark an average processor utilization of 22.6% and a frequent change of utilization in the range of 0% to 58%; energy consumption could be reduced to 12.7% of the energy required by a system running at top speed.

Plan

- 1 Introduction
- 2 State-of-the-art energy saving mechanisms
- 3 The multithreaded Komodo microcontroller core
- 4 Energy saving mechanisms using GP scheduling scheme
- 5 Evaluation
- 6 Conclusions and future work

Keywords

power-aware program execution, real-time power-management, real-time scheduling, multithreading, guaranteed percentage scheduling

1 Introduction

The reduction of energy consumption is an important research field because the number of battery-powered mobile and embedded devices is rapidly growing. Energy reduction for longer battery life concerns not only cellular phones and PDAs but also the areas of industrial control systems, future ubiquitous appliances, and sensor networks. Very simple processors and microcontrollers are used in the latter areas. Hard real-time is often an essential requirement for such systems. This paper focuses on power management in relatively simple processor cores in combination with real-time applications. The aim is to reduce the total energy consumption by optimizing power consumption without increasing the execution time of the application.

We developed a multithreaded Java microcontroller – called Komodo microcontroller – with hardware integrated real-time scheduling schemes [1, 2] for application in embedded real-time systems. The power management technique presented here is based on the Guaranteed Percentage (GP) real-time scheduling scheme [3], newly devised for multithreaded processor cores, that allows to assign percentages of the processor’s execution cycles to different threads. Instructions of active threads are executed in an overlapped fashion inside the core pipeline; the scheduler hardware ensures that the specified percentages are guaranteed within 100 clock cycles.

We investigate mechanisms to minimize energy consumption using hardware-based power management techniques that are made possible by a multithreaded processor core with integrated GP scheduling. In particular, we show that power saving techniques like frequency reduction and voltage scaling can be controlled more efficient by the integrated GP scheduler than using conventional operating system methods. With our approach, we reached a so far unrivaled level of fine-grained power management corresponding to the application’s processing requirements. Our hardware-integrated power management algorithm chooses automatically the frequency and voltage level that is currently required to perform a real-time application without any miss of deadline.

The next section shows state-of-the-art power saving mechanisms. Section 3 shortly describes the Komodo microcontroller core with integrated GP scheduling. Section 4 presents the extensions for power management inside the processor core based on the GP scheduling parameters and in section 5 we evaluate our approach. Section 6 concludes the paper.

2 State-of-the-art energy saving mechanisms

Commercial processors use a number of techniques for saving energy like pipeline gating, several suspend or sleep modes, and reduction of frequency and supply voltage. Intel's XScale and Transmeta's Crusoe processors work with a software-controlled technique of frequency reduction and voltage scaling.

Pipeline Gating is a technique for selectively disconnecting parts of the processor, especially pipeline stages [4]. So the power dissipation can be reduced by uncoupling unnecessary parts of the pipeline without concerning any other components. In contrast, frequency and voltage scaling affect the whole circuit.

Processors are enhanced with the ability of running at multiple frequencies and dynamic voltage scaling capabilities. Setting up on such processors, different directions of research are present: power management controlled by the application or the operation system. Application based power management requires special power control sequences within the application's program code. Shin et al. [5] present a technique for automatic insertion of power controlling code based on a WCET analysis before runtime. The suggested mechanism is feasible for hard real-time systems.

In contrast to application based techniques, other approaches focus on frequency and voltage reduction only controlled by the operating system, especially by its thread scheduler. Pillai et al. [6] present several new power-aware scheduling schemes for low-power embedded real-time operating systems. Jejurikar et al. [7] focus on the problem of task synchronization in combination with energy-aware task scheduling. Pouwelse et al. [8, 9] describe a hybrid approach, which is based on an extended Linux OS with a so-called *energy priority scheduling*. Parameterizing of the scheduler is done by the application.

All presented techniques are based on a single-threaded processor core and a software-based power management. The only power management investigations concerning multithreaded processors pertain simultaneous multithreading [10, 11]. The behavior of the combination of a multithreaded single-issue processor with integrated hardware real-time scheduling in regard to energy efficient program execution was to our knowledge not yet studied.

In the following we describe the energy saving mechanisms of Intel's XScale and Transmeta's Crusoe processors in more detail, because we use their electrical properties for simulating our hardware-based power

management.

The XScale is designed especially for mobile communication devices. For better support of battery-powered devices, the XScale is able to work in four different modes of activity: *turbo mode*, *run mode*, *idle mode*, and *sleep mode*. Additionally, the XScale processor is able to run at several frequencies using different supply voltages.

A change of frequency requires among other tasks to complete all outstanding memory accesses, to set the external SDRAM to self-refresh mode, and to disable the interrupt controller [12]. Most tasks are done automatically, but, nevertheless, they need time for execution. The whole process of changing frequency needs up to $500\mu\text{s}$. During this time the processor is not able to service any interrupt requests nor is the DMA controller able to handle any DMA transfer.

In contrast to the XScale processor, Transmeta's Crusoe is a general purpose processor targeting high-end mobile systems. The Crusoe supports frequency reduction and voltage scaling besides the standard *Advanced Configuration and Power Interface* (ACPI) modes. Several tasks similar to the XScale are required for switching frequency and supply voltage. The Crusoe TM5800-1000-VLP-2.1-CR80 exhibits seven different frequencies and also seven voltage levels [13]. The time required for a supply voltage change depends on the distance of the two voltage levels. The maximum value depends on the used silicon revision and is about $896\mu\text{s}$ in the default configuration.

All existing processors and research approaches suffer from the inefficiency of software control: Changing frequency and supply voltage by software requires a lot of time. A more efficient solution would be a hardware-based power management, i.e. the processor core decides to run at the optimal frequency and voltage level by itself.

In the following we present a hardware-based power management solution using a multithreaded microcontroller core with integrated real-time scheduling. The real-time parameters will be used for a hardware-driven frequency selection and an adapted supply voltage choice.

3 The multithreaded Komodo microcontroller core

Our multithreaded processor (fig. 1) consists of four pipeline stages: instruction fetch (IF), instruction decode (ID), operand fetch (OF), and execute (EX). Multithreading with four hardware thread slots is imple-

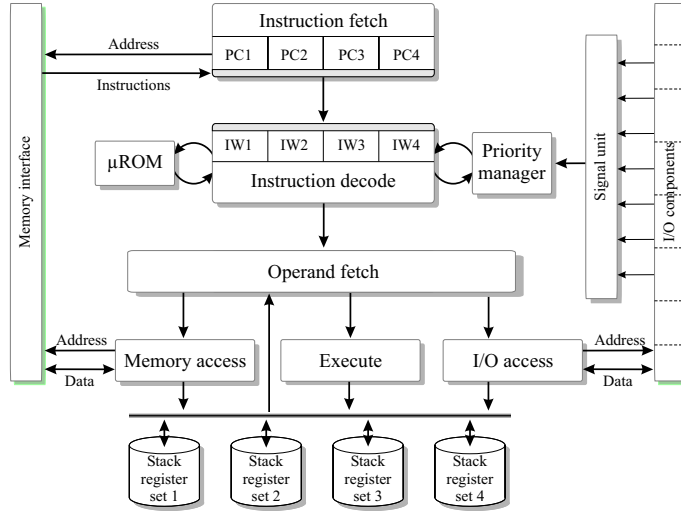


Figure 1: The Komodo processor core

mented by four program counters (PCs) and their corresponding instruction windows (IWs) in the IF stage. The IF stage fetches instructions into the IWs as soon as an IW's filling level falls below a threshold value.

The GP real-time scheduler is integrated in the ID stage within the priority manager and selects the IW from which an instruction will be decoded next. Using GP, every thread gets a predefined percentage of computing time and each thread is executed exactly the specified number of cycles within intervals of 100 execution cycles, i.e. the specified percentage. If all threads are executed corresponding to the defined percentages, the scheduler stalls the pipeline until the end of the interval is reached. Applying this procedure, the quoted percentage of computing time is guaranteed to every thread. Deadlines are fulfilled (provided that the assigned percentage is chosen not less than $\frac{WCET}{deadline}$) and the processor's performance can be adjusted to the application's requirements.

The processor model is realized in the Komodo microcontroller and described in full detail in [2, 14].

4 Energy saving mechanisms using GP scheduling scheme

The idea presented in this paper is to reduce energy consumption by using the GP real-time scheduling parameters given by the threads. A

detailed description of how the GP scheduling decision (without power management) is performed in every clock cycle is shown in [14].

With knowledge of the thread’s demand of computing time, an efficient power management is possible. The specified percentages of computing power of each thread are known within the priority manager. The overall needed performance is the sum of all active threads. If the required computing power is manifested, the system’s performance can be adjusted within intervals of 100 *base clock cycles* (i.e. clock cycles without any frequency dividing applied). Because an exact adjustment to the required frequency is not possible, additional pipeline gating is required for gating out unused pipeline cycles.

Frequency Adjustment: Frequency can be reduced depending on the overall computing requirements. We present a power management controlling a frequency divider, which allows dividing frequency by the factors of 1.5, 2, 2.5, 3, 3.5, 4, 4.5, 5, 10, and 15. Consequently, the frequency divider has to run at the highest clock rate. If frequency reduction is possible because the required overall computing power is less than 100%, a factor is selected by the microcontroller’s power management hardware, which leads to a frequency as near as possible to the required value, but not below. Frequency adjustment takes place at the beginning of every interval of about 100 base clock cycles.

Voltage scaling: If the processor core is running at lower frequency, voltage scaling is possible, too. Reducing supply voltage depends on the selected frequency and must be within the ranges given by the technical specifications of the used chip technology. In the case of an increasing demand on computing performance, first supply voltage has to be adjusted before frequency can be incremented. As shown in section 5, we used different voltage scaling characteristics derived from Intel’s XScale and Transmeta’s Crusoe processor.

Pipeline Gating: In most cases, clock frequency cannot be adjusted to the exact value needed by the software. A small number of pipeline cycles remains unused by the application. To settle this difference in an energy aware way, the dispensable pipeline cycles should be gated. Due to the architecture of the Komodo pipeline, only the 2 pipeline stages *operand fetch* and *execute/memory access/io access* can be gated. The *priority manager*, located in the *decode stage*, cannot be gated, because of its responsibility for power management and hence for the pipeline gating intrinsically. The remaining pipeline stage, the *instruction fetch stage*, is only working on demand, so gating this stage is not necessary.

By the application of these three techniques a very fine-grained frequency adjustment and furthermore voltage scaling is reached. The following pseudo-code clarifies the procedure of power management by the hardware scheduler.

```

count = 100
WHILE true
  IF count=0 THEN
    // only at the beginning
    // of an interval
    sum=0
    // calculating the required
    // overall performance
    FOR all thread slots
      IF this_thread is active THEN
        sum=sum+percentage[this_thread]
      END IF
    END FOR
    // selecting the highest possible
    // clock divider and the
    // corresponding voltage level
    SELECT
    sum < 6:
      count=6
      clock_divider=15
      voltage=lowest voltage level
    ...
    sum < 66:
      count=66
      clock_divider=1.5
      voltage=appropriate voltage level
    else:
      count=100
      clock_divider=1
      voltage=highest voltage level
    END SELECT
  END IF

  // every clock cycle
  IF thiscycles_is_needed THEN
    // making scheduling decision
    determine thread for execution
    decode instruction
  ELSE
    // otherwise: pipeline gating
    gate out pipeline stages
  END IF
  count = count - 1
WEND

```

The advantage of this power management over all other approaches is the fact that no additional power management software is required. That means, no software overhead is needed and therefore its execution does not consume any energy. Additionally, the presented hardware solution is able to react faster to changes on computing power than a

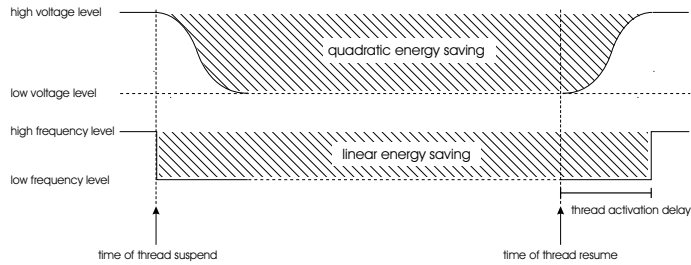


Figure 2: Relationship between frequency adjustment and voltage scaling

software solution could do.

Using the mentioned frequency divider and the method shown above, clock adjustment is possible at the beginning of every interval of 100 base clock cycles. We have to distinguish between base clock cycles and pipeline clock cycles, because the pipeline may currently run with a lower frequency than the real oscillator frequency. As seen in the pseudo code, the *count* variable is initialized at the beginning of every interval with a value depending on the clock divider. So a small divergence between the real interval length resulting from the frequency dividing and the desired length of 100 base clock cycles is possible. Consequently, the length of an interval is about 100 base clock cycles but not necessarily 100 pipeline clock cycles. If we assume for example a required performance of 66%, the divider would be 1.5 and the interval length would be 66 pipeline cycles, which results in an interval length of 99 base clock cycles.

After adjusting the pipeline frequency, reducing supply voltage is the next step. In the opposite case of an increasing performance demand, the supply voltage has to be set to the needed level before clock frequency can be adjusted. Because of the electrical capacity of the whole processor core, several hundred cycles are needed until the supply voltage reaches the required level. When increasing clock frequency before the upper voltage level is reached, malfunction of the whole circuit would occur. We managed this problem by delaying the activation of an additional thread by a value depending on the electrical properties of the processor. This procedure is not presented in the pseudo code shown above. Instead, Figure 2 demonstrates what happens during a thread's deactivation and activation when using voltage scaling and the required *thread activation delay*.

Due to the need of selecting a frequency not lower than the required performance, unused pipeline cycles can occur at the end of an interval. To minimize this waste of energy, the power management inside the priority manager is able to gate the clock signal of the two pipeline stages *operand fetch* and *execute/memory access/io access*.

Our power management techniques guarantees that all running real-time applications receive enough computing time to finish within their deadlines. If a new (real-time) thread is started, the frequency and voltage are automatically increased without any effect on the running threads. However, the newly started thread begins working after the *thread activation delay*, i.e. the *thread activation delay* has to be added systematically to the WCET of each thread.

In the next section, we present the results of simulations varying in the thread activation delay and the number of different voltage steps derived from the XScale and the Crusoe processors.

5 Evaluation

The aim of our evaluation is to prove the suitability of hardware controlled power management based on the implemented GP scheduling scheme. We developed a FPGA prototype of the Komodo microcontroller running at a pipeline frequency of 4.125 MHz. Within this prototype, the GP scheduling scheme is implemented for six available hardware thread slots. Moreover we showed in [14] that the Komodo microcontroller is able to run at frequencies of about 300 MHz using a 0.18 micron standard cell ASIC technology. When using a more recent chip technology, a much higher clock frequency should be possible. Therefore, we classify our multithreaded microcontroller architecture with hardware power management in the performance range of the XScale PXA26x and the Crusoe processors.

Benchmarking

The Komodo microcontroller prototype was built into an autonomous guided vehicle (AGV). Four hard real-time threads control the movements of the vehicle and are used here for evaluation. The microcontroller's inputs are the data sent by a line camera, its outputs are pulse width modulated signals (PWM) for two driving engines. The task of the vehicle is to track a steering line on the floor. The four threads perform the following tasks:

Thread	Percentage
Receiving camera data	25
Recognizing the line	30
Calculating steering data	3
Generating PWM signals	2

Table 1: Percentages used for the benchmark program

1. *Receiving camera data:* This thread is responsible for receiving the digital pixel values sent by the line camera. The data is stored in a Java array. The camera thread is activated each time a pixel is sent and deactivates itself after writing the received data in the array. When the whole picture is received, the array is transmitted to the second thread.
2. *Recognizing the line:* The task of this thread is to recognize the line that guides the vehicle based on the data within the array. If the line is detected, the ordinate of its center will be sent to the next thread. This thread is only active during the line detection, otherwise it is deactivated.
3. *Calculating steering data:* Together with the data of previous line pictures and the information about the actual positioning of the line, this thread calculates the new driving direction and speed. These two values are forwarded to the next thread. Thereafter, the thread deactivates itself.
4. *Generating PWM signals:* This thread’s job is to use the values of direction and speed for calculating PWM signals. The thread is only active during this task.

The four threads are scheduled by GP scheduling with the percentages shown in table 1.

Methodology

The measurement methodology (see fig. 3) combines real input data from the AGV prototype with a VHDL simulation of the Komodo microcontroller including the different power management techniques.

First, the vehicle’s control program was executed on the FPGA prototype inside the vehicle. During the first 3.2 million clock cycles, i.e. about 0.8 s, a logic analyzer records the signals sent from the line camera. The second step is to use the logged data as input to the simulation running the same vehicle program yielding the frequency and voltage changes and the number of pipeline gatings.

We made several measurements with different versions of power-

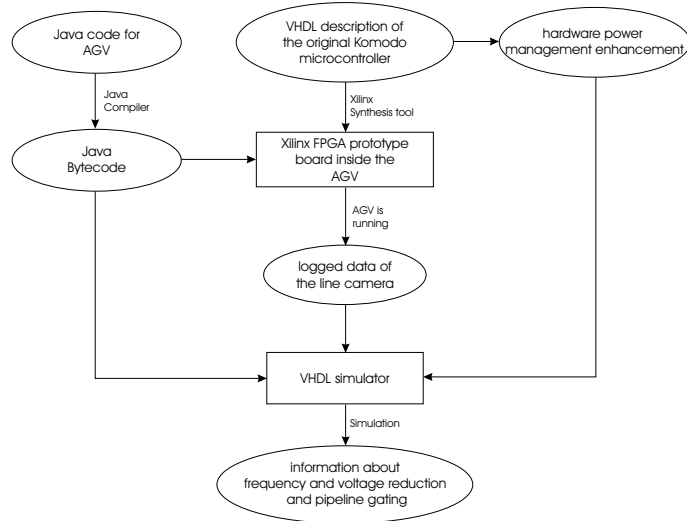


Figure 3: Benchmarking of the hardware-based power management integrated in an Autonomous Guided Vehicle (AGV)

aware program execution:

Pipeline Gating: The simplest version uses only pipeline gating, assuming an energy demand in gated mode of 30% of the energy needed when running at full speed.

Frequency Adjustment: The second version of the power-aware Komodo microcontroller uses only frequency adjustment without voltage scaling and without pipeline gating. Frequency can be changed at the beginning of every 100 base clock cycle interval.

Frequency Adjustment and Voltage Scaling: The third version uses frequency adjustment and voltage scaling.

Frequency Adjustment, Voltage Scaling, and Pipeline Gating: The fourth measurement is made using all three techniques.

Each of the four models is evaluated with supply voltages and thread activation delays derived from Intel's XScale respectively Transmeta's Crusoe processor. All eight versions of the Komodo microcontroller are simulated with the same set of input data, i.e. all measurements are made with identical processor workloads. The thread activation delay

Clock divider	XScale's voltage [V]	Crusoe's voltage [V]
1	1.1	1.3
1.5	1.0	1.05
2	1.0	0.95
2.5	1.0	0.875
3	0.85	0.85
3.5	0.85	0.8
4	0.85	0.8
4.5	0.85	0.8
5	0.85	0.8
10	0.85	0.8
15	0.85	0.8

Table 2: Voltage levels assumed for the simulation derived from the XScale and the Crusoe processor's voltage levels

for the XScale models is 2100 base clock cycles, for the Crusoe models 3700 base clock cycles. Both values depend on the time the processor needs to adjust a new voltage level. The supply voltage levels used for the simulation likewise depend on the levels used by the commercial processors. Table 2 shows the Komodo microcontroller's clock divider and the corresponding supply voltages.

Results

Figures 4-6 present the results of our simulations. The x-axis mirrors the time in base clock cycles. Figure 4 shows the percentages of the processor utilization without any power management during the first 3.2 million clock cycles. Except for the start-up code the utilization varies in the range of 0% to 58%. The visible peaks represent the periods of the active camera control thread. The four big boxes reaching the 30% mark mirror the activity of the line detection thread and the two smaller boxes demonstrate the remaining two threads. There are approximately four line pictures processed completely within the measured period of time.

Figure 5 shows the fractions of the frequency automatically selected by the Komodo microcontroller. Figure 6 presents the voltage levels using a thread activation delay derived from the Crusoe. The voltages oscillate between 0.8 and 1.05 Volt (and likewise between 0.85 and 1.0 Volt for the XScale) due to the values given in table 2.

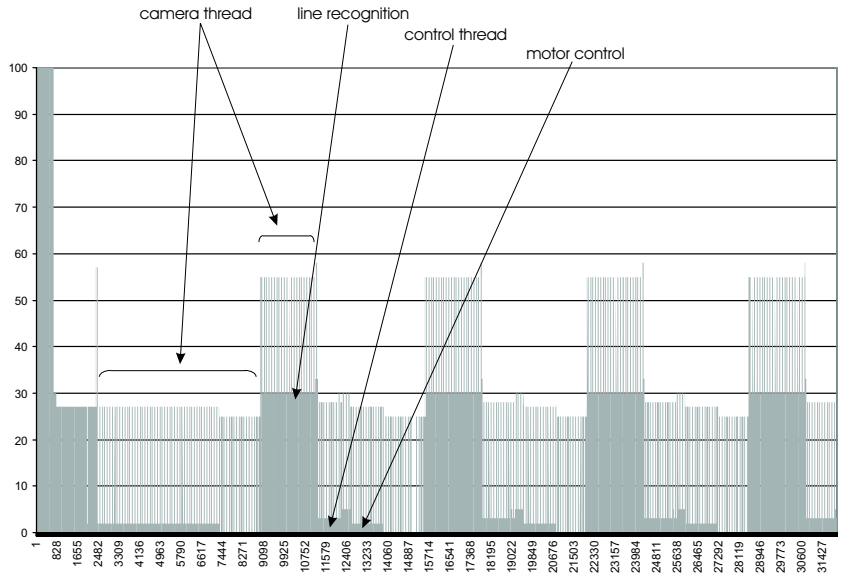


Figure 4: The utilization in percent (Y axis) without any power management

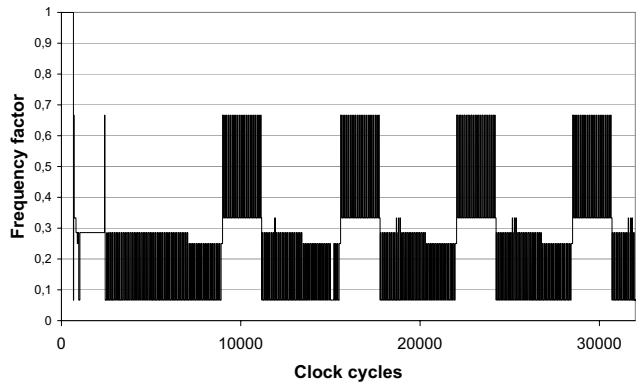


Figure 5: The fraction of the highest core frequency automatically selected by the Komodo microcontroller using the clock divider of table 2

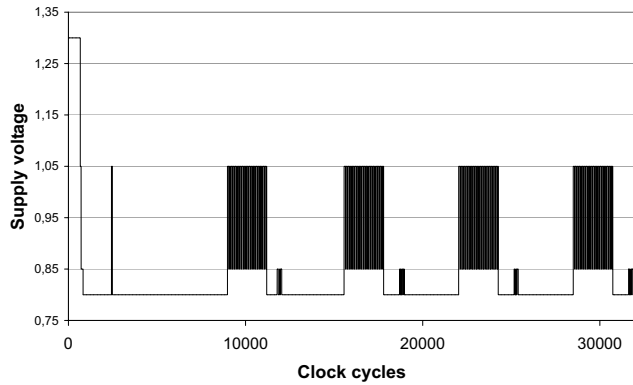


Figure 6: The Y axis shows the supply voltage automatically selected by the Komodo microcontroller using voltage levels similar to Transmeta's Crusoe processor

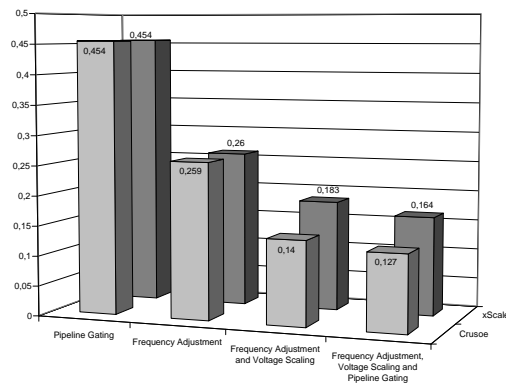


Figure 7: Total energy consumption normalized to a Komodo microcontroller running at full speed

Figure 7 summarizes the simulation results by showing the fractions of energy consumption during the simulated time interval. Each column represents the required energy in the specified technology in comparison to a Komodo microcontroller running at full speed all the time. These values are calculated using the formula

$$E = \sum(F(t) \cdot U(t)^2) \cdot C$$

where C (the capacity of the whole circuit) is normalized to 1. Frequency F and voltage U are taken from the first 3.2 million clock cycles as shown in the previous figures.

The leftmost bars show the energy consumption using pipeline gating and the highest voltage of the corresponding technology. The reason for the large energy saving of about 55% is the low overall microcontroller utilization with an average of 22.6% over the time interval. Because we assumed that the energy needed in gated mode is still 30% of the energy in running mode, the required fraction of energy (45.4%) is higher than the overall utilization.

The bars in the middle of figure 7 mirror the results with integrated hardware-based frequency adjustment and with both, frequency and supply voltage adjustment. The bars representing only frequency adjustment show values close to the overall utilization of 22.6%; the differences between the frequency adjustment values and the overall utilization are justified by the fact that an exact frequency adjustment is not possible. Using additional voltage scaling the energy consumption is extremely low in comparison to the processors utilization.

The remaining bars show the results using a combination of frequency/voltage adjustment and pipeline gating. This combination reaches the best results with the least energy consumption due to gating the unnecessary pipeline cycles that arise in the previous measurements. Because of more available voltage levels and lower voltage at the slow clock rates, the Crusoe-derived version outperforms the XScale version.

The most important point is that not only power dissipation is reduced but also the total energy consumption. This is well founded, because power dissipation is reduced during the whole runtime and the total execution time remains stable, except of the thread activation delay. These two facts lead to the reduced energy consumptions presented in figure 7.

Our results show only the ratio of energy consumption with or without the power management technique. Therefore the results are independent of the maximum clock speed of a processor, because energy consumption depends on frequency in a linear way. This means, using a faster processor core and a workload corresponding to the speed factor would lead to the same results.

6 Conclusions and future work

This paper presents a new management technique for reducing energy consumption within multithreaded real-time systems. Frequency adjustment and dynamic voltage scaling are managed exclusively by hardware. The management technique is based on the Guaranteed Percentage scheduling scheme implemented in the priority manager, which was originally devised for real-time thread scheduling available inside the multithreaded Komodo processor core. The thread execution state information present within the priority manager is used automatically by hardware for controlling frequency adjustment and dynamic voltage scaling. Pipeline gating is applied additionally. So, no software control is required for an optimal performance/energy efficiency.

Our evaluation showed that for a given workload with an average processor utilization of 22.6% and a frequent change of utilization in the range of 0% to 58% energy consumption could be reduced to 12.7% of the energy required by a system running at top speed. This value was reached with the help of very fine-grained frequency and voltage adjustment without any software overhead.

As future work, we want to explore the possibility of reducing the percentages assigned to the real-time threads. A separate optimization thread observes the progress of the real-time threads and increases/decreases the appropriate percentages just-in-time. We expect further energy savings because generally the calculated WCET is too pessimistic.

Additionally, we want to examine other real-time scheduling schemes concerning hardware-based power-management, especially the Earliest Deadline First (EDF) scheduling. Within our Komodo microcontroller EDF scheduling is already implemented and we plan to design and evaluate a hardware-based EDF power management.

References

- [1] J. Kreuzinger, A. Schulz, M. Pfeffer, T. Ungerer, U. Brinkschulte, and C. Krakowski, "Real-time Scheduling on Multithreaded Processors," in *7th International Conference on Real-Time Computing Systems and Applications (RTCSA 2000)*, Cheju Island, South Korea, Dec. 2000, pp. 155–159.
- [2] J. Kreuzinger, U. Brinkschulte, M. Pfeffer, S. Uhrig, and T. Ungerer, "Real-time event-handling and scheduling on a multithreaded Java Microcontroller," *Microprocessors and Microsystems*, vol. 27, no. 1, pp. 19–31, Feb. 2003.
- [3] U. Brinkschulte, J. Kreuzinger, M. Pfeffer, and T. Ungerer, "A Scheduling Technique Providing a Strict Isolation of Real-time Threads," in *Seventh IEEE International Workshop on Object-oriented Real-time Dependable Systems (WORDS)*, San Diego, CA, USA, Jan. 2002, pp. 334–340.
- [4] H. Li, S. Bhunia, Y. Chen, T. N. Vijaykumar, and K. Roy, "Deterministic clock gating to reduce microprocessor power," in *International Symposium on High-Performance Computer Architecture (HPCA)*, Feb. 2003, pp. 113–122.
- [5] D. Shin, J. Kim, and S. Lee, "Intra-task voltage scheduling for low-energy hard real-time applications," *IEEE Design and Test of Computers*, vol. 18, no. 2, Mar./Apr. 2001.
- [6] P. Pillai and K. G. Shin, "Real-time dynamic voltage scaling for low-power embedded operating systems," in *ACM Symposium on Operating Systems Principles*, 2001, pp. 89–102.
- [7] R. Jejurikar and R. Gupta, "Energy aware task scheduling with task synchronization for embedded real time systems," in *International Conference on Compilers, Architectures and Synthesis for Embedded Systems*, Grenoble, France, 2002, pp. 164–169.
- [8] J. Pouwelse, K. Langendoen, and H. Sips, "Energy priority scheduling for variable voltage processors," in *Int. Symposium on Low Power Electronics and Design (ISLPED)*, Huntington Beach, CA, USA, Aug. 2001.

- [9] J. Pouwelse, K. Langendoen, and H. Sips, "Dynamic voltage scaling on a low-power microprocessor," in *7th ACM International Conference on Mobile Computing and Networking (Mobicom), Rome, Italy*, July 2001, pp. 251–259.
- [10] D. Brooks, P. Bose, S. Schuster, H. Jacobson, P. Kudva, A. Buyuktosunoglu, J.-D. Wellman, V. Zyuban, M. Gupta, and P. Cook, "Power-aware Microarchitecture: Designing and Modeling Challenges for Next-generation Microprocessors," vol. 20, no. 6, pp. 26–44, Nov./Dec. 2000.
- [11] J. Seng, D. Tullsen, and G. Cai, "Power-sensitive multithreaded architecture," in *2000 IEEE International Conference on Computer Design: VLSI in Computers and Processors, Austin, TX, USA, 2000*, pp. 199–206.
- [12] *Intel PXA26x Processor Family Developer's Manual*, Intel Corporation, Oct. 2002.
- [13] *Crusoe TM5500/TM5800 System Design Guide*, Transmeta Corporation, July 2002.
- [14] S. Uhrig, C. Liemke, M. Pfeffer, J. Becker, U. Brinkschulte, and T. Ungerer, "Implementing real-time scheduling within a multi-threaded java microcontroller," in *35th International Symposium on Microarchitecture (MICRO-35), 6th Workshop on Multithreaded Execution, Architecture and Compilation (MTEAC-6), Istanbul, Turkey*, Nov. 2002, pp. 57–64.