Low-Energy Real-Time OS Using Voltage Scheduling Algorithm for Variable Voltage Processors

Okuma, Takanori
Department of Computer Science and Communication Engineering

Yasuura, Hiroto
Department of Computer Science and Communication Engineering

出版情報：The 10th Workshop on System And System Integration of Mixed Technologies(SASIMI 2001), pp.340-345, 2001-10. Workshop on System And System Integration of Mixed Technologies
バージョン：
権利関係：

Low-Energy Real-Time OS Using Voltage Scheduling Algorithm for Variable Voltage Processors

Takanori Okuma Hiroto Yasuura
Department of Computer Science and Communication Engineering
Graduate School of Information Science and Electrical Engineering
Kyushu University
6–1 Kasuga-koen, Kasuga, 816-8580 Japan

Abstract— This paper presents a real-time OS based on μITRON using proposed voltage scheduling algorithm for variable voltage processors which can vary supply voltage dynamically. The proposed voltage scheduling algorithms assign voltage level for each task dynamically in order to minimize energy consumption under timing constraints. Using the presented real-time OS, running tasks with low supply voltage leads to drastic energy reduction. In addition, the presented voltage scheduling algorithm is evaluated by the built real-time OS.

I. Introduction

The demand for systems which can perform various processes for a long time with the limited battery is increasing by the spread of cellular phones or PDAs(Personal Digital Assistant). Thus, design technology for high-performance system-on-chips (SOCs) with low energy consumption is an important research issue. Increasing clock frequency of a processor contributes to high-performance systems, but it dissipates more energy. Therefore, there is a trade-off between clock frequency and energy consumption in processor-based core systems.

The effective ways to reduce energy consumption of a processor in CMOS technology are to shutdown the supply voltage when the system is not operated[12] and to lower constantly the supply voltage level, which exploits the quadratic dependence of energy on voltage[3]. However, to lower constantly the supply voltage level is usable only if the system having loose deadline requirements. A single tight latency constraint, as is often present in embedded systems, renders the technique ineffective. In recent years, variable voltage processors which can vary their supply voltage dynamically are presented[1, 2]. Then, the clock frequency is adjusted to the supply voltage, to guarantee the correct operations. Using the variable voltage processor, tasks with severe real-time constraints can be executed with a high supply voltage (therefore, high execution speed), and tasks with loose time constraints are done with a low supply voltage. For various applications, performance requirements of a processor are different. It makes possible to lower energy consumption using the difference of performance requirements. In addition, the special instructions of the processor for controlling the supply voltage makes possible to control the supply voltage by the software applications[7]. Then, the supply voltage of the processor is controlled by application programs or operating systems. Therefore, it is important to establish compiler and operating system techniques which control the supply voltage for energy minimization of real-time systems.

We have presented voltage scheduling algorithms which assign voltage level for each task in [13]. This algorithm determines not only the executed task but also supply voltage for processors. Then, the supply voltage for the minimize energy is selected under the time constraints of the whole tasks.

In this paper, we developed a real-time OS using the presented voltage scheduling algorithm[13] and evaluated the voltage scheduling algorithm.

The rest of the paper is organized in the following way. Section 2 presents the related work. Section 3 explains voltage scheduling algorithm. Section 4 describes the presented real-time OS using voltage scheduling algorithm and Section 5 gives experiments and results. Section 6 concludes our works.

II. Related Work

Lee and Sakurai have proposed a runtime dynamic voltage-scaling scheme for low-power real-time systems[8, 9]. This scheme employs a power control chip with an on-chip DC-to-DC converter and frequency synthesizer, as well as an embedded runtime power control algorithm using the software feedback loop. The scheme avoids interface problems by exploiting discrete levels of clock frequency as $f_{CLK}, f_{CLK}/2, f_{CLK}/3, ...$, where f_{CLK} is the master (highest) system clock frequency.

Hong et al. describe a design methodology for a real-time system on a chip that uses a dynamically variable voltage processor core. This methodology provides an offline scheduling heuristic to handle nonpreemptive, hard real-time tasks and select the processor core. It also de-

termines the configuration and size of the instruction and data caches[5]. Also, Hong and other colleagues have proposed an online preemptive scheduling algorithm for on- and offline tasks on a variable voltage processor to optimize energy consumption while ensuring that all offline tasks meet their deadlines. They also designed the algorithm to accept the highest possible number of online tasks that can be guaranteed to meet their deadlines[6]

Shin et al. proposed a power-efficient version of fixed-priority preemptive scheduling, which is widely used in hard real-time system design[11]. Their method reduces energy consumption in the processor by exploiting system inherent slack times, as well as slack times arising from dynamic variations of execution times for the task.

Pering et al. presented an online scheduling algorithm for soft real-time systems[10]. This algorithm relaxes the deadline constraints and allows application frames to complete after their deadlines. The scheduler can then absorb the effects of high frame-to-frame application variance, which might otherwise increase energy.

Burd et al. have demonstrated dynamic voltage scaling on a complete embedded processor system[2]. This prototype system contains four custom chips in $0.6\mu m$ three-metal CMOS: a battery-powered DC-to-DC voltage converter, a microprocessor (ARM8 core with 16-Kbyte cache), SRAM memory chips, and an interface chip for connecting to commercial I/O devices. The entire system can operate from 1.2 to $3.8V$ and $580MHz$, and energy consumption varies from 0.54 to $5.6mW/MIP$.

III. Voltage Scheduling Algorithm

A. Processor Model

To control supply voltage dynamically, a processor which has special instruction is assumed. A program can vary supply voltage by using this instruction. Fig.1 shows the architecture of a variable voltage system for our processor model. The processor has limit voltage levels. Therefore, the program can't use continuous voltage levels. The processor has the table of processor modes in which the supply voltage and the clock frequency are made as one pair. The program should select a processor mode by referencing the table when it wants to change supply voltage. In addition, we consider single processor system.

For variable voltage systems, the time and power overhead to change the supply voltage sometimes be an important issue. However, this paper ignores these overhead. Because an extension of problem considering these overhead is not difficult, and this issue has already discussed in [8].

B. Task Model

In many real-time control applications, periodic activities represent the major computational demand in the system. Such applications consist of several concurrent periodic tasks with individual timing constraints. The operating system has to guarantee that each periodic instance is regularly activated at its proper rate and is completed within its deadline. In this paper, we consider a set of periodic tasks. To facilitate the description of the task model, the following notation is introduced:

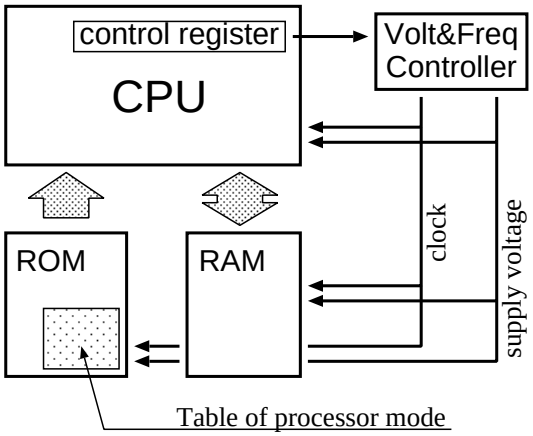


Fig. 1. Architecture of a Variable Voltage System

- $\mathcal{T}$ denotes a set of periodic tasks.
- T_i denotes a generic periodic task.
- $T_{i,j}$ denotes the jth instance of task T_i.
- $r_{i,j}$ denotes the release time of the jth instance of task T_i.
- $d_{i,j}$ denotes the absolute deadline of the jth instance of task T_i.

The instances of a periodic task T_i are regularly activated at a constant rate. The interval p_i between two consecutive activations is the period of the task. All instances of a periodic task T_i have the same worst case execution cycles WC_i and relative deadline D_i which is equal to the period p_i. In addition, no task can suspend itself, for example on I/O operations. The release time $r_{i,j}$ and the absolute deadline $d_{i,j}$ of the generic jth instance can then be computed as

$$\begin{aligned} r_{i,j} &= (j-1)\cdot p_i \\ d_{i,j} &= r_{i,j} + p_i = j \cdot p_i \end{aligned}$$

C. Application Program Model

An application program is structured by an operating system and some user tasks (periodic tasks). When an application program changes supply voltages, it determines an processor mode by referencing clock frequencies not by referencing voltage levels in processor mode table because timing constraints are more important than energy

constraints. In addition, User tasks can't use special instruction which controls supply voltage because it is a privileged instruction. Only an operating system can use this instruction.

D. Energy Model

The energy consumption per clock cycle for a task is shown by the following expression.

$$E_{cycle} = \sum_{k=1}^{M} LC_k \cdot SW_k \cdot {V_{DD}}^2 \quad (1)$$

where M is the number of gates in the circuit, LC_k is the load capacitance of a gate g_k, SW_k is the switching count of g_k per clock cycle for the task, and V_{DD} is the supply voltage. Let us consider a task with the number of total execution cycles EC_i. The energy consumption for the task is formulated as (2).

$$E_i = EC_i \cdot \sum_{k=1}^{M} LC_k \cdot SW_k \cdot {V_{DD}}^2 \quad (2)$$

We can reduce the energy consumption for the task by lowering V_{DD}. However, lowering V_{DD} causes increase of execution time for the task. The circuit delay τ and execution time for the task ET_i can be formulated as (3), and (4), respectively.

$$\tau \propto \frac{V_{DD}}{(V_G - V_T)^\alpha} \sim \frac{1}{V_{DD}} \quad (1 < \alpha \leq 2) \quad (3)$$

$$ET_i = \tau \cdot WC_i \quad (4)$$

where V_T is the threshold voltage, and V_G ($\sim V_{DD}$) is the voltage of input gate. The α is a factor depending on the carrier velocity saturation and is about 1.3 in advanced MOSFETs.

E. Scheduling Algorithm

In the presented voltage-scheduling algorithm, it is assumed that the scheduler assigns a supply voltage to only the next executed task just before task execution. Then, the scheduler must assign supply voltage so that all tasks executed later will not violate these real-time constraints.

We define a time slot for each task. The start time of a time slot is the same as the start time of the task execution. The end time is the maximum time which can guarantee that all future executed tasks will not violate these real-time constraints. Thus, if the next executed task's supply voltage lets it finish within the time slot that the scheduler gives it, satisfaction of the real-time constraints is always guaranteed.

In our techniques, the scheduler's main work is to determine the time slot's length for each task. The remained scheduler's work is to assign the minimum voltage to the task so that it can finish within its time slot.

The presented voltage scheduling algorithm has three main steps:

Step1: CPU time allocation. Assign CPU time to the task set under the condition that all tasks execute on V_{max} and that the execution cycle for each task is the worst case. This step is statically phase.

Step2: End-time prediction. Determine the time slot's end time for all tasks, considering real-time constraints of all later executed tasks. This step is statically phase too.

Step3: Start-time assignment. Determine the time slot's start time. The end time of the previously executed task dynamically moves to the start time; the time slot can be lengthened if the previous task finishes ahead of schedule. This step is dynamically phase.

IV. Low-Energy Real-Time OS

We built a real-time OS based on μITRON using the presented voltage scheduling algorithm.

A. Target Architecture

The target architecture of the real-time OS is the follows:

- Instruction set is same as SH3 (Hitachi SuperH).
- The number of processor modes are m. The program can use following frequencies to change the clock frequency.
$$f, \frac{f}{2}, \frac{f}{3}, \cdots, \frac{f}{m} \ \text{[Hz]}$$
- The supply voltage changes automatically with the clock frequency.
- The processor permits interrupt by interval timer. If an interrupt handler is registered, interrupt can execute the handler.

When the scheduler selects the processor mode, it determines the value of m for clock frequency $\frac{f}{m}$.

B. System Clock

The interval timer handler counts the variable which manages the time used by real-time OS. The variable is called system clock. The time unit of real-time OS is the period of interval timer. An interrupt handler is executed after every N clock cycles. The period of interval timer changes with the processor clock frequency. Another variable is necessary for real-time OS to express real executed time. The variable is called real clock. When the clock is changed to $\frac{f}{m}$, the period of interval timer is m times of the unit time. When system clock counted as 1 unit by interval timer hander, it is possible to express executed time if real clock counted as m by interval timer hander.

real_clock: Variable of real clock

sys_clock: Variable of system clock

N: cycle count during the period of system clock

$\mathcal{P}$: a set of being preempted tasks (static variable)

$T_{i,j}$: jth instance of task T_i

$s_{i,j}$: execution start time of task $T_{i,j}$

$e_{i,j}$: predicted end time of task $T_{i,j}$

$w_{i,j}$: remaining worst case real clock time of task $T_{i,j}$ (initial: $w_{i,j} = \lceil \frac{WC_i}{N} \rceil$)

p_time: sum of w_i (static variable)

Input:
$T_{i,j}(s_{i,j}, e_{i,j}, w_{i,j})$: executed task by now
$T_{k,l}(s_{k,l}, e_{k,l}, w_{k,l})$: next executed task

Algorithm:

```
1)  if T_{i,j} is preempted then
2)    w_{i,j} = w_{i,j} − (sys_clock − s_{i,j})
3)    Insert T_{i,j} to P
4)  end if
5)
6)  if T_{k,l} ∈ P then
7)    Remove T_{k,l} from P
8)    if P is empty then
9)      e_{k,l} = e_{k,l} + p_time
10)     p_time = 0
11)   end if
12) else
13)   if P is not empty then
14)     p_time = p_time + w_{k,l}
15)   end if
16) end if
17)
18) if P is empty then
19)   m = ⌊ (e_{k,l} − real_clock) / w_{k,l} ⌋
20)   if m < 1 then
21)     m = 1
22)   end if
23) else
24)   m = 1
25) end if
26) s_{k,l} = real_clock
27) return m
```

Fig. 2. The Algorithm for Determining m

C. Implementing the Scheduling Algorithm

When it switches tasks, the scheduler determines the next task and the value of m for clock frequency $\frac{f}{m}$ of the processor. Then, the scheduler sends the instruction which changes the clock frequency of the processor, and move on the next task. The algorithm for determining m is shown in Fig.2. It is called when a task is switched. In this algorithm, it determines the value of m for execution task $T_{k,l}$ when switching from $T_{i,j}$ to $T_{k,l}$.

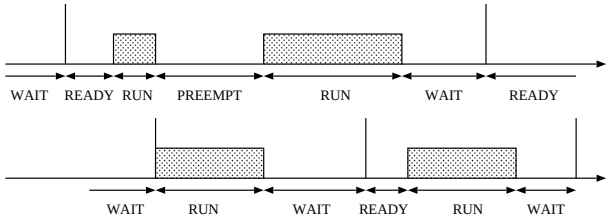


Fig. 3. An Example of Task State Transition

The predicted end time for each task $e_{i,j}$ is computed statically. Then, it is set the value of real clock (the unit time for real-time OS) corresponding real time. This algorithm determines the scheduling based on the real clock. Just like what we described in the previous section, $e_{i,j}$ must be set so as to satisfy real-time constraints of all tasks which will be executed after $T_{i,j}$, where all cases should be considered.

Normally, the next executed task is assigned the maximum value of m for completion of the task from current time to predicted end time by the scheduler (Fig.2: line 19). If the executed task is preempted, the value $m = 1$ is assigned to any other tasks until the task is executed again (Fig.2: line 24). When the preempted task is executed again, the predicted end time is updated (Fig.2: line 9) and the preempted task is assigned the maximum value of m for completion of the task from current time to the predicted end time.

The states of task include four states, which are READY, RUN, PREEMPT and WAIT. The READY is the state that the instance of a task is waiting for its execution since it was released. The RUN is the state that it's running. The PREEMPT is the state that it's waiting for continuation of its execution since it was preempted. Finally, the WAIT is the state that it's waiting for its release of next instance since its execution was finished. An example of task state transition is shown in Fig.3. When all task are WAIT state, that is CPU idle, the system is shutdown during the CPU idle for no energy consumption.

V. Experiments

We evaluated the presented voltage scheduling algorithm by building a real-time OS. This section presents our experiments and results.

A. Emulator for Variable Voltage Processor

In order to run the real-time OS, it is necessary to use a hardware system or a software emulator for variable voltage processors. We had built an emulator to conduct our experiments. The emulator can run the our real-time OS, satisfy the target architecture for the presented scheduling

TABLE I
TASK SET FOR EXPERIMENTS

(unit: real clock cycle)

Parameter		T_0	T_1
Period:	p_i	10	14
WCEC:	$w_i = \lceil \frac{WC_i}{N} \rceil$	4	5
		$N = 1000$	

algorithm and evaluate the timing and energy constraints. The developed emulator has the following functions:

- Simulation on instruction level.
- Interrupt by interval timer
- Dynamic variation of clock frequency
- Profiling of execution trace

In addition, we use GNU C Compiler (gcc) as a compiler for the application program.

B. Experimental Results

In experiments, we verified timing constraints and evaluated energy consumption using several task sets on the presented real-time OS. We consider task set which is shown in TABLE I. In the table, the value of each parameters for T_0 and T_1 is shown. Because our scheduling algorithm is designed based on the real clock, the time unit of tasks is real clock cycle in the task set. In addition, the period of interrupt by interval timer N (the period of system clock) is 1000 clock cycles. The set of periodic tasks is schedulable with fixed voltage because the following condition[4] is satisfy.

$$\sum_{i=0}^{n-1} \frac{w_i}{p_i} \leq 1 \tag{5}$$

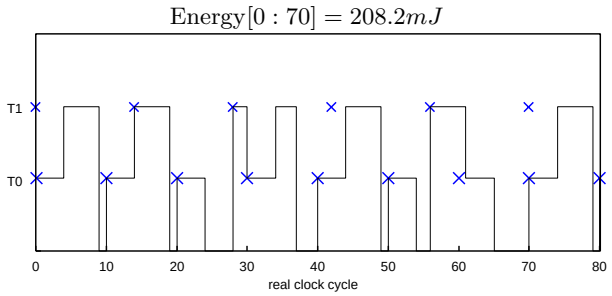
This task set and our built real-time OS are executed on the emulator. Fig.4 shows analysis results of profile information which was generated by the emulator.

(a) The task set is executed with fixed voltage (always $m = 1$) and actual execution cycles of each task is worst case.

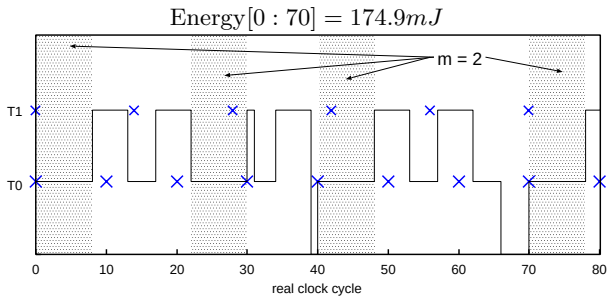
(b) The task set is executed with variable voltage and actual execution cycles of each task is worst case.

(c) The task set is executed with fixed voltage (always $m = 1$) and actual execution cycles of each task is about half of worst case.

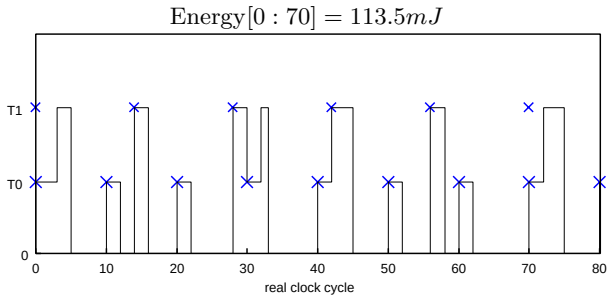
(d) The task set is executed with variable voltage and actual execution cycles of each task is about half of worst case.



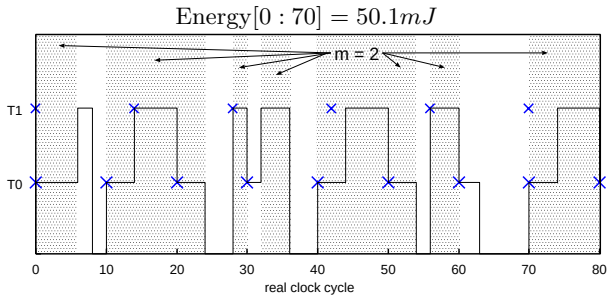
(a) Fixed Voltage and Worst Case Execution Cycle



(b) Variable Voltage and Worst Case Execution Cycle



(c) Fixed Voltage and WCEC/2



(d) Variable Voltage and WCEC/2

Fig. 4. Execution Results of Task Set

In Fig.4(b) and Fig.4(d) (the case of execution with variable voltage), parts of lowering voltage level are expressed shadow region. Fig.4 shows that our proposed algorithm is right behavior.

We assume that energy consumption per instruction cycle is $4\mu J$ at $m = 1$ and $1\mu J$ at $m = 2$. Fig.4 shows the

total energy consumption from 0 to 70 cycles (Energy[0 : 70]). We can achieve 16% energy saving in the case that actual execution cycles of each task is worst case and 56% energy saving in the case that actual execution cycles of each task is about half worst case.

VI. Conclusion

In this paper, we built a real-time OS based on μITRON using the presented voltage scheduling algorithm for real time variable voltage processor based embedded system. Our experimental results show that the presented real-time OS using the voltage scheduling algorithm can achieve low energy consumption under timing constraints for real time embedded system. A new voltage scheduling algorithm considering capacitance of task is our future work.

References

[1] http://www.transmeta.com/crusoe/.

[2] T. Burd, T. Pering, A. Stratakos, and R. Brodersen. "A Dynamic Voltage Scaled Microprocessor System". In *Proc. of IEEE International Solid-State Circuits Conference*, pages 294–295, 2000.

[3] A. P. Chandrakasan, S. Sheng, and R. W. Broderson. "Low-power CMOS digital design". *IEEE Journal of Solid-State Circuits*, 27(4):473–484, 1992.

[4] C.L. Liu and J.W. Layland. "Scheduling algorithms for multiprogramming in a hard-real-time environment". *Journal of the Association for Computing Machinery*, 20(1), 1973.

[5] I. Hong, D. Kirovski, G. Qu, M. Potkonjak, and M. Srivastava. "Power optimization of variable voltage core-based systems". In *Proc. of 35th Design Automation Conference*, pages 176–181, 1998.

[6] Inki Hong, Miodrag Potkonjak, and Mani B. Srivastava. "On-Line Scheduling of Hard Real-Time Tasks on Variable Voltage Processor". In *Proc. of ICCAD-98*, pages 653–656, 1998.

[7] T. Ishihara and H. Yasuura. "Voltage Scheduling Problem for Dynamically Variable Voltage Processors". In *Proc. of International Symposium on Low Power Electronics and Design(ISPLED'98)*, pages 197–202, August 1998.

[8] S. Lee and T. Sakurai. "Run-time Power Control Scheme using Software Feedback Loop for Low-power Real-time Application". In *Proc. of Asia and South Pacific Design Automation Conference 2000*, pages 381–386, 2000.

[9] S. Lee and T. Sakurai. "Run-time Voltage Hopping for Low-power Real-time Systems". In *Proc. of 37th Design Automation Conference*, pages 806–809, 2000.

[10] T. Pering, T. Burd, and R. Brodersen. "Voltage Scheduling in the lpARM Microprocessor System". In *Proc. of International Symposium on Low Power Electronics and Design(ISPLED'00)*, pages 96–101, 2000.

[11] Y. Shin and K. Choi. "Power Conscious Fixed Priority Scheduling for Hard Real-Time Systems". In *Proc. of 36th Design Automation Conference*, pages 134–139, 1999.

[12] M. Srivastava, A. P. Chandrakasan, and R. W. Brodersen. "Predictive System Shutdown and Other Architectural Techniques for Energy Efficient Programmable Computation". *IEEE Transactions on VLSI Systems*, 4(1):42–55, 1996.

[13] T. Okuma, T. Ishihara, and H. Yasuura. "Real-Time Task Scheduling for a Variable Voltage Processor". In *Proc. of 12th International Symposium on System Synthesis*, pages 25–29, 1999.