社団法人 情報処理学会 研究報告
IPSJ SIG Technical Report

2007−ARC−171 (10)
2007−EMB− 3 (10)
2007／1／22

Power Management for Idle Time in the Presence of Periodic Interrupt Services

Gang Zeng Hiroyuki Tomiyama Hiroaki Takada
Graduate School of Information Science, Nagoya University, Japan
E-mail: {sogo, tomiyama, hiro}@ertl.jp

Abstract

Generally, there are periodic interrupt services in the real-time embedded systems even when the system is in the idle state, such as periodic clock tick interrupts. To obtain the minimal idle power, power management therefore should consider the effect of periodic interrupt services. In this paper, we deal with this problem under different conditions. In the case that the periodic interrupt cannot be disabled, we first model its power consumption and then propose static and dynamic approaches to select the optimal frequency for minimal power consumption. On the other hand, if the periodic interrupt can be disabled, we propose an approach to delay the interrupt service until the next task is released so that the processor can stay in low power mode for longer time. The proposed approaches are implemented on a real-time OS and its effectiveness has been validated by theoretical calculations and actually measurements on an embedded processor.

1. Introduction

Energy consumption has become one of the major concerns in today's embedded system design especially for battery-powered devices. For the sake of dependability, in real-time systems the utilization of processor is less than 100% even when all tasks run at WCET (worse case execution time). Moreover, workload of each task may vary from time to time, which results in the less average execution time than the WCET. All these factors lead to the system idle state at which there is no tasks needed to be scheduled. It should be noted that even in the idle state, most real-time OS maintains a periodic clock interrupt to synchronize the system and trace the clock events. For example the uc/OS-II, eCOS, and Linux need a 10ms clock interrupt to generate the system clock. Besides the period clock tick, some interrupt-driven embedded systems such as data acquisition systems also need periodic interrupts to activate the CPU from low power mode for data processing. To reduce the power of the idle state, a common approach is to transfer the processor into a low power mode which consumes less power than the normal mode. Generally, a processor can provide multiple low power modes to deal with different system states. To take advantages of these power control mechanisms, dynamic power management (DPM) tries to assign the optimal low power mode according to the predicted duration of the system idle state. As for examples, Figure 1 shows the power mode transition graph for two typical embedded processors in high-end and low-end applications, respectively.

While the SA-1100 with integrated 32-bit RISC core targets for high performance low power application, the M16C with integrated 16-bit core, on-chip ROM and RAM aims at low-end and low power application. The SA-1100 processor provides three operation modes with different power consumption levels, i.e., Run, Idle, and Sleep modes. While the Run mode is the normal operating mode with full functionalities and high power consumption, the Idle and Sleep modes are low power mode with stopped CPU clock. Idle mode stops the CPU core clock but enables all peripherals clock thus on- or off-chip interrupt service requests can quickly reactivate the CPU. In contrast, Sleep mode stops both CPU and peripherals clock thus only hardware reset or special event can wakeup the CPU, which requires long transition time whenever entering or exiting the sleep mode. Similarly, the M16C also provides three power modes which have similar functionalities to that of the SA-1100 but with different names. For example, the wait mode is similar to idle mode, and has disabled CPU clock and enabled peripheral clocks. The stop mode stops both the CPU and peripherals clocks, which is similar to the sleep mode of SA-1100. However, it should be noted that the time and power overhead of M16C for power mode transition is much less than that of SA-1100. This small transition overhead of M16C is benefited from its simple and single-chip architecture. Actually, only one instruction is needed to transfer the processor into idle mode.

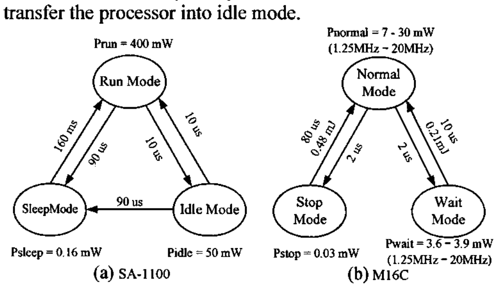


Fig. 1. Power mode transition for (a) Intel's StrongARM SA-1100 processor [1] (b) Renesas's M16C processor.

Although the Sleep mode has the lowest power consumption, it is not suitable for the application considered in this paper. The reasons are that (1) the transition time overhead for returning to run mode is too large to be used in the application with short period of interrupt services; (2) the normal interrupt service requests related to on-chip clock cannot work properly in the Sleep mode. Therefore, the feasible low power mode that can be used in the application with periodic interrupt services is the idle mode.

In addition to DPM, another effective technique for power saving is dynamic voltage/frequency scaling (DVFS), because the power consumption of CMOS circuits is proportional to its clock frequency and its voltage square. The DVFS tries to change the clock frequency and its corresponding supply voltage dynamically to the lowest possible level while meeting the task's deadline constraint. Commonly, the voltage and frequency scaling are accomplished by controlling a DC-DC converter and PLL (phase lock loop), respectively. While many high-end processors have equipped with the DVFS capabilities, few low-end processors can dynamically change their supply voltages such as the M16C. In contrast, most low-end processors can also change its clock frequency by setting the divider registers. As a result, the time overhead for frequency change is much less for a simple processor using divider register than a complex processor using PLL. For example, while M16C requires negligible time for frequency change, Intel's PXA255 processor (an upgraded product of SA-1100 series) requires about 500us [7]. For simplicity, we refer to DVFS in the following whenever voltage and frequency or only frequency is changed during execution.

The motivation of this research is originated from the fact that the power consumed in idle mode is not fixed but dependent on the selected clock frequency before entering the idle mode [7]. Generally, the higher frequency, the more power is consumed in idle mode. For example, the PXA225 processor consumes 45mW-121mW power in idle mode which corresponds to 100MHz - 400MHz frequency, respectively [7]. The reason is that although the disabled CPU cannot consume dynamic power in idle mode, the enabled peripherals still consume power which is directly dependent on the selected clock frequency [8]. To reduce the power of idle mode we therefore expect to lower the frequency. However the lowered frequency will lead to longer execution time for interrupt service routine (ISR), which may result in higher power consumption in execution state. Accordingly, we need to determine the optimal frequency for the idle state with period interrupt services to save power. In this paper, we propose corresponding approaches for idle state power management under different conditions. In the case that the periodic interrupt cannot be disabled, static and dynamic approaches are proposed to select the optimal frequency for power savings. In contrast, if the periodic interrupt can be disabled such as the clock tick interrupt, a configurable clock tick is proposed to save power in idle state.

The rest of the paper is organized as follows. Section 2 gives related work. Section 3 presents the power model and the proposed approaches. In Section 4, experimental results are described. Finally, Section 5 summarizes the paper.

2. Related work

Recently, there have been a large number of publications using DPM or DVFS for power savings. Most DPM literatures focus on the design of power management policies based on predictive schemes and stochastic optimum control schemes [1][4]. To the best of our knowledge, no paper considers the low power mode with varied power consumption which is dependent on the selected clock frequency. They generally assume fixed power consumption for each low power mode. The different policies are to decide when and which low power mode the devices should transfer into. Actually, an on-chip timer interrupt is commonly employed in embedded systems to reactivate the CPU from low power mode. In this case, the on-chip clock cannot be disabled. Moreover, the returning time from low power mode with all clock disabled will be longer than that of low power mode with enabled peripheral clock.

While DPM aims to the long idle time, and save power by entering low power mode; DVFS aims to short slack time generated among the running of multiple tasks, and save power by lowering the voltage and/or frequency. DVFS algorithms generally assume periodic tasks with known WCET and deadline. Although the objective of DVFS is to prolong the task execution time to deadline by lowering the CPU's voltage and frequency, the slack time cannot be reclaimed completely. This is because the generated slack time can only be reclaimed when there is ready task that can be scheduled. Moreover, the discrete frequency levels makes DVFS cannot utilize the generated slack time completely. All these factors result in idle time even in the DVFS enabled systems. However, most DVFS literatures ignore the idle time process by simply assuming zero power or fixed power consumption for idle time [2][3]. As discussed in Section 1, even in low power mode, the power consumption is neither zero nor fixed value, moreover the required low power mode transition time may be too long to be applicable for short slack time.

Recently, a variable scheduling timeouts method is proposed for power savings in Linux systems by eliminating the useless tick interrupts during system idle time [9]. However a problem that has to be considered in real-time systems is the system clock synchronization caused by the stopping and restart of tick timer.

3. Power model and proposed approaches

For general low power embedded processors, we assume the processor can provide multiple low power modes and alterable voltage/frequency for power control. To simply the calculation, we assume the time and power overhead for power mode transition and voltage/frequency scaling are fixed. As discussed earlier, for power management of idle state with periodic interrupt services, only the low power mode with enabled peripherals clock is considered. We assume that an idle task is employed to implement the proposed power management in RTOS. The idle task is scheduled to run, when system enters the idle state in which no tasks need to be scheduled in the ready queue.

We deal with the power saving problem of idle state in two cases. While in case one the periodic interrupt cannot be disabled such as the data acquisition system, in case two the interrupt can be disabled for a specified period such as clock tick interrupt.

3.1 Case one: the periodic interrupt cannot be disabled

Before modeling the power consumption of idle state with periodic interrupt services, we give the following notations.

- M: selected system speed, i.e., 1/M full speed
- T_p (us): period of interrupt service
- T_h (us): execution time of interrupt service routine at full speed
- T_s (us): execution time for low power mode setting in idle task at full speed
- T_t (us): time overhead for power mode transition
- I_t (mA): average current during power mode transition
- I_{rm} (mA): the run mode average current at 1/M full speed
- I_{im} (mA): the idle mode average current at 1/M full speed

Considering the fact that different scale processors may have different DVFS overhead as discussed in Section 1, we propose static and dynamic methods for processors with large or slight DVFS overhead, respectively.

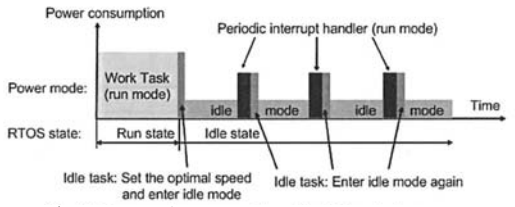

Power consumption
Periodic interrupt handler (run mode)
Power mode:
Work Task (run mode)
idle
mode
idle
mode
Time
RTOS state:
Run state
Idle state
Idle task: Set the optimal speed and enter idle mode
Idle task: Enter idle mode again

Fig.2. Processing procedure for idle state power management.

If the processor has large DVFS time overhead, a static approach is adopted, i.e., only once DVFS setting at the beginning of idle state for any continuous idle time. Specifically, the program in idle task takes corresponding actions based on the current state of system, if it is the first time to enter idle state, it first setups the optimal speed for power savings and then enters low power mode. Otherwise, it only setups and enters the low power mode and without any speed change when the idle task is reactivated from low power mode by interrupt. The above processing procedure for idle power management is illustrated in Fig.2. Based on the above notations and procedure, the average current of idle state with periodic interrupt services can be represented by the following equation:

$$I_{idle}=\frac{(T_h+T_s)\cdot M\cdot I_{rm}+(T_p-(T_h+T_s)\cdot M-T_t)\cdot I_{im}+T_t\cdot I_t}{T_p}. \quad (1)$$

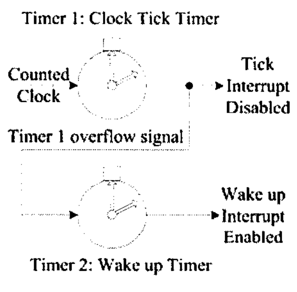
Therefore, if the period of interrupt and the execution time for power mode setting are fixed and known, time and power overhead for power mode transition, the average current with different speed settings for run and idle mode can be obtained from manual or measurement, the average current of idle state will be a function of the selected speed M and the execution time of interrupt service T_h.

According to the above function, the power optimization problem can be formulated as: for a specified processor and application with known T_h, T_s, T_t, I_t, I_{rm}, and I_{im}, finds the optimal M such that the average idle current is minimal. Because the relation between I_{idle} and M is linear, and the selectable speeds are limited, we can calculate all curves of I_{idle}-T_h with all possible speed selections, and then the one that has the minimal average current will be the optimal speed setting.

If the processor has slight DVFS time overhead, a dynamic approach may save more power at the expense of two DVFS settings for each interrupt. The processing procedure is that the fastest speed is set at the beginning of each interrupt service, and the slowest speed is set before entering the low power mode each time. Its objective is to save more power by keeping the processor in low power mode for longer time with the minimal power consumption. Note that this approach is not realistic for some complex processors with large DVFS overhead. For example, Intel's PXA225 requires 500us for each DVFS scaling [7], thus it is not applicable for the interrupt service with 1 ms period.

3.2 Case two: the periodic interrupt can be disabled for a specified period

We assume periodic tasks with known WCET and deadline in embedded systems, and we only discuss how to disable clock tick interrupt by using a configurable clock tick in order to save more power during idle state. Under the above assumptions, whenever system detects the beginning of an idle state, it also knows the nearest releasing time of a periodic task. In this case, the duration of the idle state is known, we therefore can disable the clock tick for this known idle time and transfer the processor to low power mode to save power for the idle time. Note that this approach is different from general DPM in that while general DPM makes decision for power mode transition based on the predicted duration of idle time; this approach is with known duration of idle time. Therefore the decision for power mode transition in this approach is straightforward.



Timer setting procedure:
When entering the idle state
1. Set the count value of Timer2 as the known duration of idle state
2. Disable Timer 1 interrupt
3. Start Timer 2
4. Enter low power mode

When returning to normal mode by Timer 2 interrupt
1. Update the system tick by adding the loss clock ticks to current clock ticks
2. STOP Timer 2
3. Enable Timer 1 interrupt

Fig. 3. Configurable clock tick and timer setting procedure.

When the clock tick interrupt is disabled during idle state, a problem to be considered is how to trace the original clock tick to keep system time synchronization. To this end, another timer, as shown in Fig.3, can be used to count the lost ticks during idle time when the tick interrupt is disabled. Because the original tick timer is never stopped and restarted except disabling its interrupt request, the system time synchronization can be realized easily. However, this approach is hardware-dependent since a connection wire between the output of timer 1 and the input of timer 2 is required as shown in Fig.3. The count value of timer 2 for generating the wakeup interrupt prior to the release of next task should be set to the known duration of idle state. The detailed timer setting procedure is listed in Fig.3.

4. Evaluation and experimental results

4.1 Experiment setup and measurement environment

To validate and evaluate the proposed approach, we select the Renesas'M16C embedded processor to implement the approach. Although the processor cannot change its supply voltage, it provides three power modes and can quickly change its clock frequencies by setting the divider registers. We measure the processor current by inserting a digital tester between the power supply and the power pin of the processor. An oscilloscope is utilized to observe the voltage waveform of the shunt resistant which is inserted between the power supply and the power pin of the processor. The time and power overhead for power mode transition can be estimated from the captured voltage waveform. Note that the above experiments are performed separately so that the current measurements are carried out with removed shunt resistant. The employed experiment environment is shown in Fig.4, and the measured power results and estimated power mode transition overhead have been given in Fig.1.

Our approach has been implemented on a RTOS called TOPPERS/JSP kernel [5] which is an open source RTOS in consistent with the uITRON [6] standard. The TOPPERS RTOS targets for real-time applications with limited resource requirement. A configurable clock tick is implemented on OS with default 1 ms interrupt period. The normal execution time of the timer handler for system time updating is about 12 us at 20MHz.

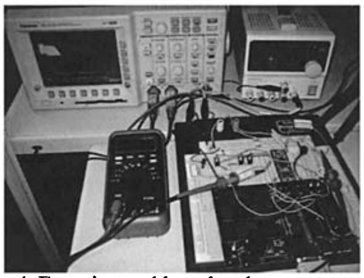


Fig. 4. Experimental board and measurement environment.

4.2 Evaluation of the proposed approach when the periodic interrupts cannot be disabled

Table. 1 summaries the measured normal and wait mode average current under different speed settings. Note that all these measurements are performed by executing a busy loop and the results for wait mode is measured with clock enable but without any interrupt services.

Based on these measured parameters, and equation (1), we can obtain the following current vs. time and speed curves in Fig.5. From this figure, it is clear that the optimal speed selection for minimal power consumption is determined by the execution time of timer handler. As for the 12us interrupt service in this experiment, the optimal speed is 10MHz (1/2 full speed). The calculated and measured results are denoted in Table 2, respectively where the optimal measured result with minimal power consumption is consistent with the theoretical calculated results. We reduce the interrupt service time to 7us, and measure the average current for different speed settings again. As can be seen from Table 3, 5MHz (1/4 full speed) can achieve the minimal power consumption, which is also consistent with the calculated results.

Table 1. Measured normal and wait mode average current under different speed settings.

Selectable Speeds (1/M full speed)	Measured current (mA)(voltage = 3V)	
	Normal mode: I_{rm}	Wait mode: I_{im}
20MHz (1/1)	10.04	1.30
10MHz (1/2)	6.35	1.26
5MHz (1/4)	4.35	1.24
2.5MHz (1/8)	3.24	1.23
1.25MHz (1/16)	2.45	1.22

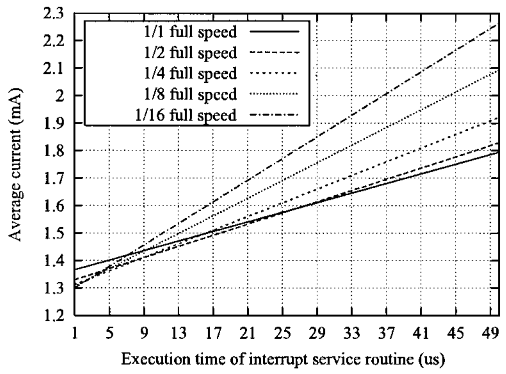


Fig. 5. Calculated results for 1ms interrupt period: average current vs. execution time and speed selection.

Table 2. Comparison of measured and calculated average current with Tp=1ms Th=12us.

Selected Speed (1/M full speed)	Idle state average current (mA) under periodic interrupt service (voltage=3V, period=1ms, Th=12us)	
	Measured current	Calculated current
20MHz (1/1)	1.47	1.472
10MHz (1/2)	**1.45**	**1.451**
5MHz (1/4)	1.47	1.461
2.5MHz (1/8)	1.50	1.498
1.25MHz (1/16)	1.57	1.534

Table 3. Comparison of measured and calculated average current with Tp=1ms Th=7us.

Selected Speed (1/M full speed)	Idle state average current (mA) under periodic interrupt service (voltage = 3V, period = 1ms, Th = 7us)	
	Measured current	Calculated current
20MHz (1/1)	1.40	1.419
10MHz (1/2)	1.38	1.389
5MHz (1/4)	**1.37**	**1.385**
2.5MHz (1/8)	1.38	1.401
1.25MHz (1/16)	1.42	1.416

We change the interrupt period to 10ms and perform the above calculations and measurements again. The corresponding results are given in Fig.6 and Table 4. As can be seen, the optimal speed is 1.25MHz (1/16full speed) for this long interrupt period. When we further prolong the interrupt period to 100ms, as can be seen in Fig.7, the slowest speed will achieve the minimal power consumption in spite of the variation of execution time. The reason is that for longer interrupt period, most of time the processor stays in low power mode, thus, the average power is dominated by the power of long idle state but not the power of short execution state.

Table 4. Comparison of measured and calculated average current with Tp=10ms Th=12us.

Selected Speed (1/M full speed)	Idle state average current (mA) under periodic interrupt service (voltage=3V, period=10ms, Th=12us)	
	Measured current	Calculated current
20MHz (1/1)	1.32	1.317
10MHz (1/2)	1.28	1.279
5MHz (1/4)	1.25	1.262
2.5MHz (1/8)	1.24	1.256
1.25MHz (1/16)	**1.24**	**1.251**

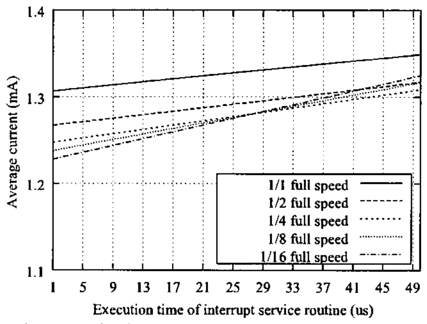


Fig. 6. Calculated results for 10ms interrupt period: average current vs. execution time and speed selection.

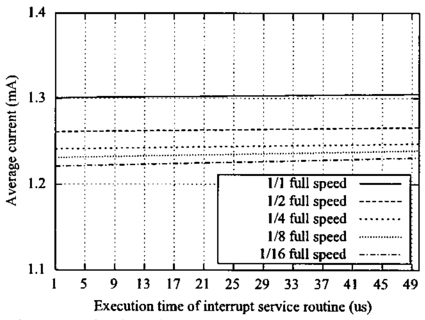


Fig. 7. Calculated results for 100ms interrupt period: average current vs. execution time and speed selection.

Experiments are also performed to validate the proposed dynamic approach especially for the M16C with negligible DVFS overhead. In these experiments, the varied speeds are set at the beginning of ISR, and the slowest speed (1/16 full speed) is set in the idle task before entering the low power mode. The calculated results using equation (1) are depicted in Fig.8 where the curves for static and dynamic approaches are displayed, respectively. As can be seen, the fastest speed setting for ISR plus the slowest speed setting for low power mode outperforms other speed combinations in dynamic approach, and all speed settings in static approach. Meanwhile, the actually measured result for this case shows average current 1.39 mA which is the minimal current compared with the measured results for static approach in Table 2. Therefore, for the processor with negligible DVFS overhead, the dynamic approach can achieve more power savings than the static approach.

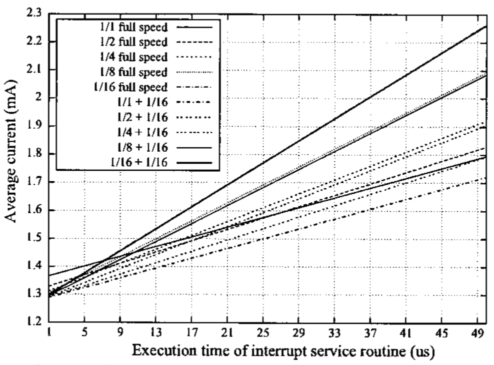


Fig.8. Calculated results for 1ms interrupt period: the static approach and dynamic approach.

4.3 Evaluation of the proposed approach when clock tick interrupts can be disabled

We evaluate the proposed configurable clock tick on the TOPPERS/JSP kernel and M16C board. Pillai and Shin 's "Cycle-conserving DVS for EDF scheduler"[2] is selected and implemented on RTOS. The experiment test set is presented in Table 5, and corresponding energy results in one minute for different idle state processing schemes are summarized in Table 6. As can be seen, while DVFS can achieve significant power savings compared with full speed running, the proposed configurable clock tick for idle state power management can further reduce the energy by 23% in average compared with normal DVFS without any idle state process.

5. Conclusion

Even in a DVFS enabled embedded system, there must be idle time. Moreover, a periodic interrupt services may be required even in the system idle state. In this work we presented different approaches for idle state power management in the presence of periodic interrupt services. In the case that the periodic interrupt cannot be disabled, we model the power consumption and propose static and dynamic methods to achieve minimal power consumption for the processors with large or slight DVFS overhead, respectively. In the case that the periodic interrupt can be disabled such as the periodic clock tick interrupt, we proposed a configurable clock tick to save power by keeping the processor in low power mode for longer time. We implement the proposed approaches on a RTOS and embedded processor; the calculated and measured results validate the effectiveness of the approaches.

Table 5. Experiment task set.

Task set	Period (ms)	WCET (ms)	Actual ET (ms)
Task 1	500-2000	130	28-130
Task 2	500-3000	245	38-245

Table 6. Evaluation of power savings for combined DVFS and power management of idle state.

Dynamic EDF DVFS with different idle state process	P1: 500 P2: 500 (ms)	P1: 500 P2: 900 (ms)	P1: 1000 P2: 1500 (ms)	P1: 2000 P2: 3000 (ms)
No DVFS (full speed)	TE: 1807 NR: 1	TE: 1807 NR: 1	TE: 1807 NR: 1	TE: 1807 NR: 1
DVFS without idle state process	TE: 1594 NR: 0.88	TE: 1288 NR: 0.71	TE: 897 NR: 0.50	TE: 468 NR: 0.26
DVFS setting the lowest speed and entering wait mode	TE: 944 NR: 0.52	TE: 773 NR: 0.43	TE: 553 NR: 0.31	TE: 282 NR: 0.16

Note: P: period; TE: Total Energy (mJ); NR: Normalized result

Acknowledgments

This work is supported by the Core Research for Evolutional Science and Technology (CREST) from Japan Science and Technology Agency.

References

[1] L. Benini, A. Bogliolo, and G. D. Micheli, "A survey of Design Techniques for System-Level Dynamic Power Management," *IEEE Transactions on Very Large Scale Integration Systems (VLSI)*, Vol. 8, No. 3, pp.299-316, June 2000.

[2] P. Pillai and K. G. Shin, "Real-Time Dynamic Voltage Scaling for Low-Power Embedded Operating Systems," *Operating Systems Review*, 35(5), pp.89-102, 2001.

[3] W. Kim, D. Shin, H. Yun, J. Kim, and S. L. Min, "Performance Comparison of Dynamic Voltage Scaling Algorithms for Hard Real-Time Systems," *Proc. IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pp.219-228, 2002.

[4] Z. Ren, B. H. Krogh, and R. Marculescu, "Hierarchical adaptive dynamic power management," *IEEE Trans. on Computers*, Vol. 54, No. 4, pp. 409-420, April 2005.

[5] TOPPERS Project, Available: http://www.toppers.jp/

[6] ITRON Project, Available: http://www.sakamura-lab.org /TRON/ITRON/

[7] Intel, Application Note "PXA255 and PXA26x Applications Processors Power Consumption During Power-up, Sleep, and Idle," April, 2003.

[8] Texas Instruments, Application Report, SPRA164, "Calculation of TMS320LC54x Power Dissipation," June 1997.

[9] Variable scheduling timeouts (VST) project page: http://tree.celinuxforum.org/CelfPubWiki/VariableSchedulingTimeouts